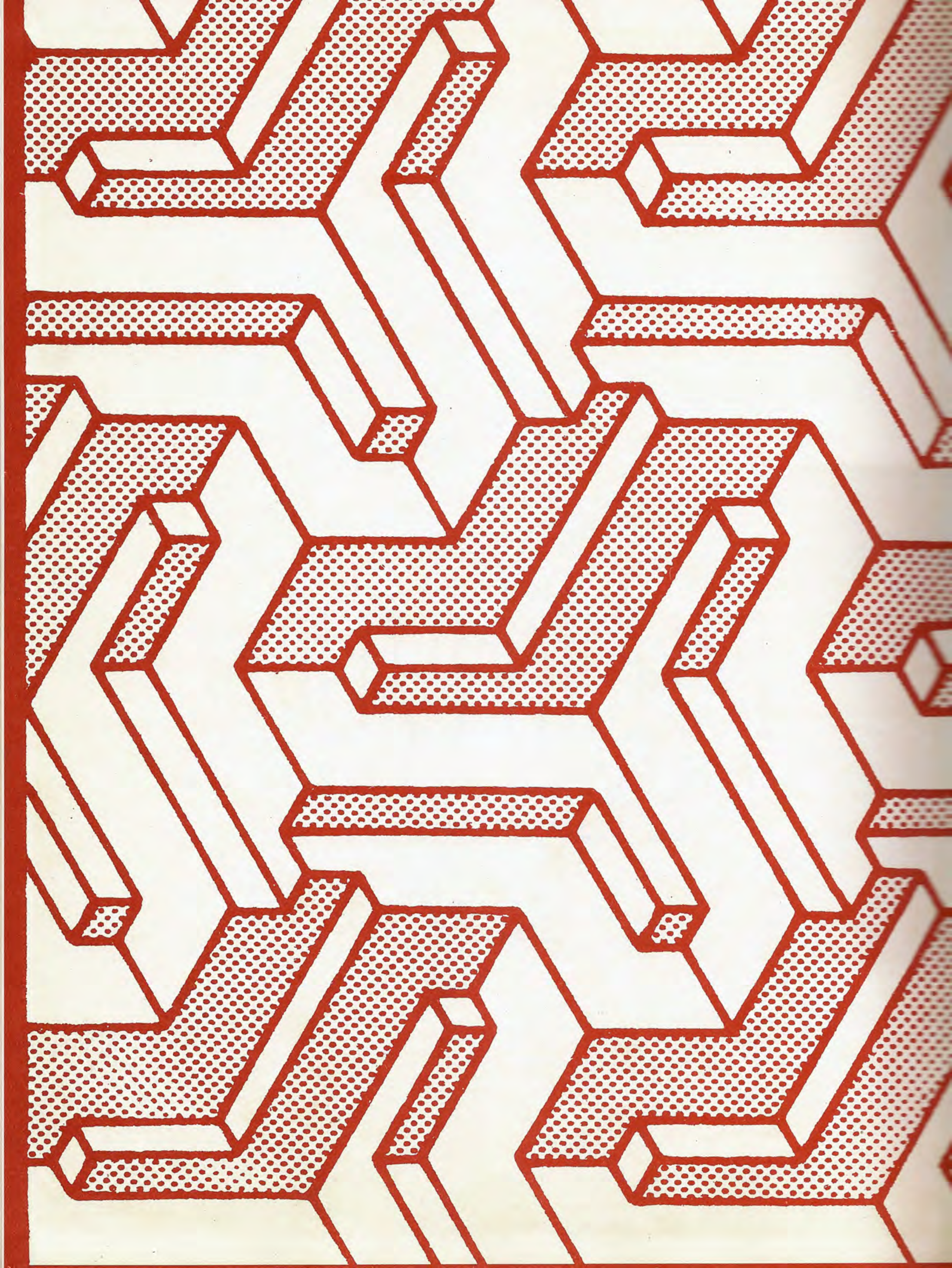
GRAN ENCICLOPEDIA INFORMATICA

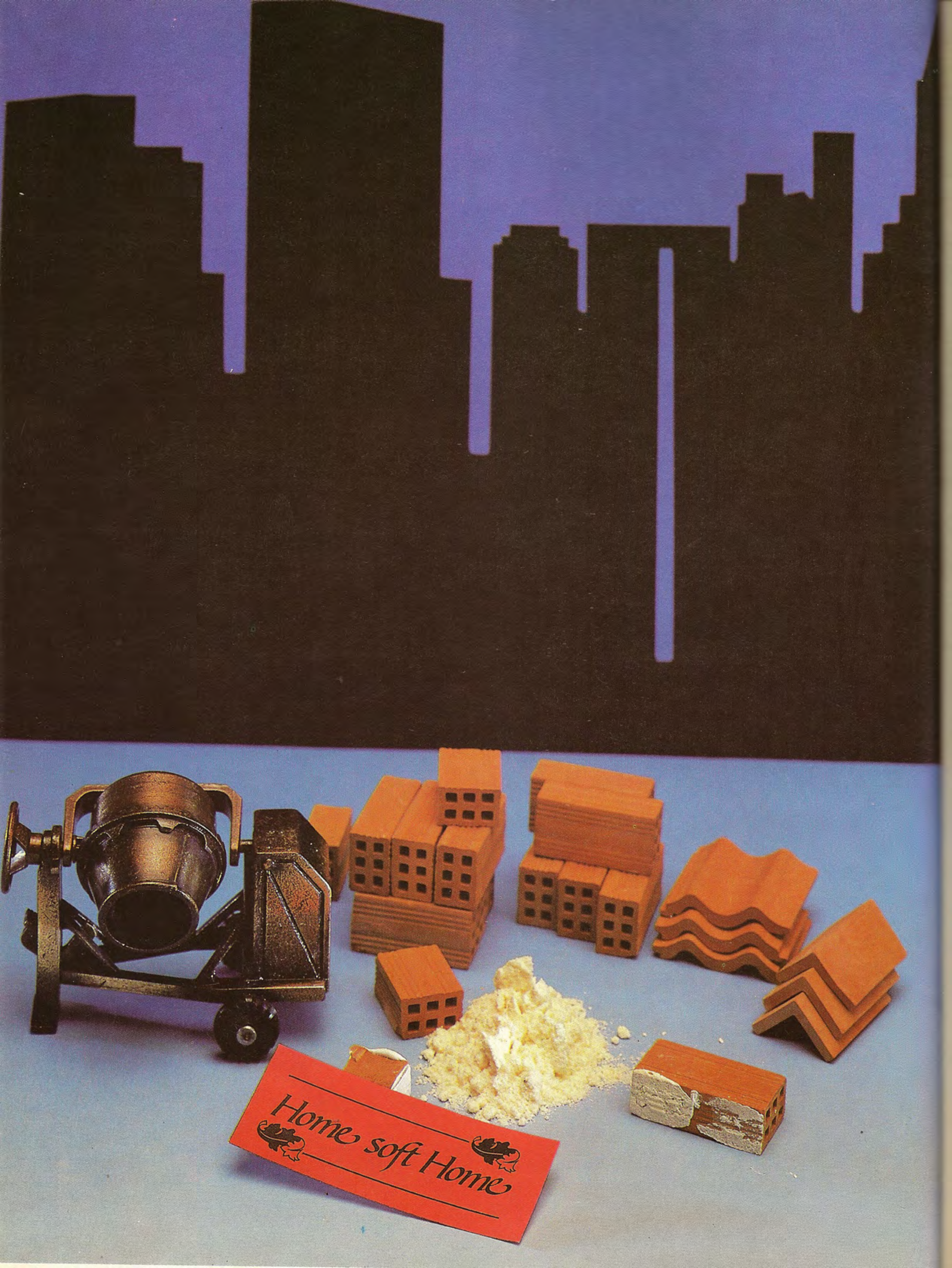
CURSO DE BASIC/ 2

EDICIONES NUEVA LENTE



GRAN ENCICLOPEDIA INFORMATICA

EDICIONES NUEVA LENTE


Home soft Home

SUMARIO

Una publicación:

Ediciones Nueva Lente, S. A.

Director editor: MIGUEL J. GOÑI

Director de producción: SANTOS ROBLES.

Director de la obra: FRANCISCO LARA.

Colaboradores: PL/3 - MANUEL MUÑOZ - ANGEL MARTINEZ - MIGUEL DE ROSENDO - DAVID SANTAOLALLA - SANTIAGO RUIZ - LUIS COCA - MIGUEL ANGEL VILA - MIGUEL ANGEL SANCHEZ VICENTE ROBLES.

Diseño: BRAVO/LOFISH.

Maquetación: JUAN JOSE DIAZ SANCHEZ.

Ilustración: JOSE OCHOA.

Fotografía: (Equipo Gálata) ALBINO LOPEZ y EDUARDO AGUDELO.

Ediciones Nueva Lente, S. A.:
Dirección y Administración:
Benito Castro, 12. 28028 Madrid. Tel.: 245 45 98.

Números atrasados y suscripciones:
Ediciones Ingelek, S. A.
Plaza de la Rep. Ecuador, 2 - 1.°. 28016 Madrid.
Tel.: 250 58 20.

Plan general de la obra:
18 tomos monográficos de aparición quincenal.

Distribución en España:
COEDIS, S. A. Valencia, 245. Tel.: 215 70 97.
08007 Barcelona.

Delegación en Madrid:
Serrano, 165. Tel.: 411 11 48.

Distribución en Argentina:
Capital: AYERBE
Interior: DGP

Distribución en Chile: Alfa Ltda.

Distribución en México:
INTERMEX, S. A.
Lucio Blanco, 435
México D.F.

Distribución en Uruguay:
Ledian, S. A.

Edita para Chile:
PYESA
Doctor Barros Borgoño, 123
Santiago de Chile

Importador exclusivo Cono Sur:
CADE, SRL. Pasaje Sud América, 1532.
Tel.: 21 24 64. Buenos Aires - 1.290. Argentina.

Fotomecánica: Ochoa, S. A.
Miguel Yuste, 32. 28037 Madrid.

Impresión: Gráficas Reunidas, S. A.
Avda. de Aragón, 56. 28027 Madrid.

ISBN de la obra: 84-7534-184-5.
ISBN del tomo 9: 84-7534-206-X

Printed in Spain
Depósito legal: M. 27.605-1986

Precio de venta al público en Canarias, Ceuta y Melilla: 940 ptas.
Enero 1987

Un día en las carreras

Un sencillo programa para repasar conceptos



niciaremos nuestra andadura por este segundo tomo dedicado al lenguaje BASIC proponiendo la realización de un pequeño programa que sirva para afianzar los tópicos tratados en el volumen anterior.

Se trata de elaborar un programa de apuestas hípicas. Las galopadas de los caballos de carreras estarán presentes en el «hipódromo de la pantalla», logrando así una completa ambientación. El caballo ganador de la contienda se dejará a elección del azar. De esta forma, el resultado será totalmente imprevisible para el apostante. Este podrá, si tiene suerte, hacerse millonario (imaginariamente, claro), o bien sumirse en la más deplorable de las ruinas.

Para ir calentando «grupas», se comenzará con el diseño del cartel de entrada. Este servirá, además, para dar a conocer las reglas del juego. Su confección no tiene mayor dificultad que la de tratar de darle una presentación atractiva y, por ende, más eficaz. Este cartel de entrada puede realizarse por medio de la siguiente rutina:



El BASIC no es más que un lenguaje destinado a facilitar la comunicación; desde luego, su diferencia respecto a los lenguajes humanos reside en que su objetivo es permitir el diálogo con las máquinas programables: los ordenadores.
Su práctica puede resultar amena, e incluso divertida, tomando como elemento de ejercicio la confección de un programa de tipo lúdico.

```
10 REM TURF
20 CLS
30 PRINT AT 14,1 ;"***TURF***"
40 PRINT AT 0,6 ;"DISPONE INICIALMENTE DE 4.000
   PTS."
50 PRINT :PRINT "LA APUESTA MAXIMA ES DE 1.500
   PTS."
60 PRINT :PRINT "SI GANA EL CABALLO APOSTADO, SE"
70 PRINT "MULTIPLICARA POR TRES LA CANTIDAD JU
   GADA"
80 PRINT AT 2,23; "PULSE UNA TECLA PARA COMEN
   ZAR";
90 LET B$=INKEY$ :IF B$=""THEN GOTO 90
```

A la hora de confeccionar este programa, se ha supuesto que la presentación en pantalla es de 24 filas (de la 0 a la 23) por 40 columnas (de la 0 a la 39).

En el caso particular de que el ordenador utilizado no comparta dicho tamaño de pantalla, habría que acomodar el cartel de entrada al número de filas y columnas disponibles. Esto mismo afectará también a la presentación en pantalla del resto del programa. Por ello es aconsejable consultar previamente el manual que acompaña a cada equipo.

Utilizando los comandos PRINT y PRINT AT, se sitúan los diversos textos en su lugar correspondiente, para que queden centrados y produzcan una presentación agradable.

Al final del segmento de programa que se encarga de crear el cartel de entrada, se encuentra una instrucción que hace uso de la función INKEY$ y una sentencia IF/THEN (línea 90). A primera vista, la referida línea de programa parece totalmente ajena al objetivo del cartel de entrada. Su cometido no es más que el de producir una espera por parte del ordenador, para dar tiempo a leer el cartel de entrada. De otra forma, el cartel desaparecería casi instantáneamente sin cumplir su cometido.

Para evitarlo, mediante el comando INKEY$ se comprueba si se ha pulsado alguna tecla. Si es así, proseguirá la ejecución del programa en la línea siguiente (que aún no ha sido escrita). De lo contrario, se volverá a comprobar si se ha pulsado ya una tecla (GOTO 90). Así pues, el segmento principal del programa no empezará a ejecutarse hasta que lo ordene el usuario; orden que introducirá pulsando una tecla cualquiera.

El resto del programa se puede dividir en tres segmentos con entidad propia. Una primera parte indicará la cantidad de dinero de la que se dispone, y pedirá la que se desea apostar, así como el caballo al que va dirigida la apuesta. La segunda parte se encargará de simular la carrera de caballos a lo largo de la pantalla. Por último, en la tercera parte se comprobará si el caballo ganador coincide con el apostado, para así actualizar la «bolsa» del apostante.

Hagan juego, señores

La primera parte del núcleo del programa se ocupará, como ya se ha indi-





El juego permite apostar una cantidad de dinero cuyo límite máximo se determina dentro del programa; dinero que el jugador puede perder o triplicar según el resultado de la carrera.

cado, de recoger la cantidad a apostar y el número del caballo en el que se va a invertir. Pero antes de ello habrá que inicializar una variable (D) con la cantidad de dinero disponible para apuestas. Esta será, como se indicó en el cartel de entrada, de 4.000 ptas. Dicha acción se efectúa en la línea 100 que contiene, además, el dimensionado que las matrices X y C; matrices que se utilizarán más adelante para memorizar las coordenadas actuales de cada uno de los caballos en carrera.

El listado de esta primera parte es el siguiente:

```
100 DIM X(5):DIM C(5):LET D=4000
110 CLS:IF D=0 THEN PRINT "NO QUEDA DINERO PARA
    APOSTAR":END
120 PRINT "TIENE ";D;" PTS."
130 PRINT
140 INPUT "CUANTO APUESTA ";A
150 IF A>0 AND A<=1500 AND A<=D THEN GOTO 180
160 PRINT "APUESTA NO VALIDA"
170 GOTO 130
180 PRINT
190 INPUT "(DEL 1 AL 5) A QUE CABALLO ";C
200 IF C<1 OR C>5 THEN GOTO 180
210 LET D=D—A
```

La línea 110 borra la pantalla con CLS, para dejarla en condiciones de ser reutilizada en esta primera zona del programa. De esta forma, nada habrá que estorbe o distraiga la concentración del apostante.

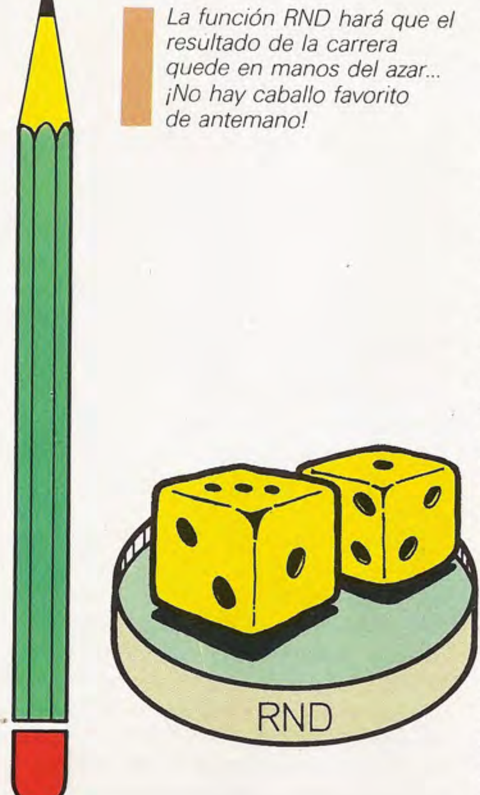
Como quiera que esta zona debe ejecutarse de nuevo cada vez que concluya una carrera, sea cual fuere el resultado de la misma, se ha incluido en la propia línea 110 una comprobación de la cantidad que posee el apostante. El programa finalizará si de esta comprobación resulta que la cantidad de dinero disponible es igual a cero... ¡el jugador se ha arruinado!

La línea 120 indicará en cada pasada la cantidad actual de que dispone el apostante. Más adelante, la línea 140 pide, mediante un comando INPUT, la cantidad que se va a apostar en la próxima carrera; cantidad que se almacenará en la variable A.

En la línea siguiente se comprueba si la cantidad es correcta. Para ello debe tratarse de un número mayor que cero (A>0), menor o igual que la apuesta máxima, que es de 1.500 ptas. ($A \leqslant 1.500$) y menor o igual que la cantidad que posee actualmente el apostante ($A \leqslant D$). Esta comprobación se realiza agrupando las condiciones antes citadas en una sola, en torno al operador lógico AND. Así, pues, deben cumplirse todas ellas para que esta nueva expresión lógica sea cierta (valor lógico 1). La mencionada expresión se utiliza para decidir, dentro de la instrucción IF/THEN de la línea 150, qué acción hay que emprender: saltar a la línea 180 del programa, si es cierta la expresión, o bien presentar el mensaje de "APUESTA NO VALIDA" y volver a la línea 130 para pedir una nueva apuesta. Esto último, en el caso de que la expresión adoptara el valor lógico 0 (falso). De forma totalmente análoga, se pedirá el número del caballo al que se dirige la apuesta. Dicho valor se asignará a la variable C, en la lí-



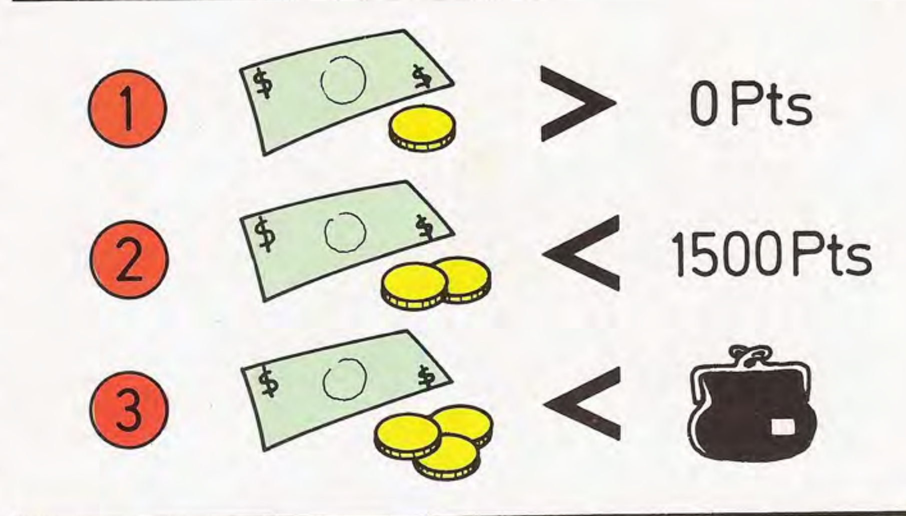
La función RND hará que el resultado de la carrera quede en manos del azar... ¡No hay caballo favorito de antemano!



nea 190. A continuación se comprueba, en la línea 200, si el valor introducido es correcto. En este caso se emplea el operador lógico OR para construir la expresión de control de la instrucción IF/THEN. Esta adoptará el valor lógico 1 cuando C sea menor que 1, o cuando sea mayor que 5 (caballos no válidos), siendo entonces cierta la condición y ejecutándose la zona THEN. Así se envía el flujo de ejecución del programa a la línea 180, en la que se pide de nuevo un valor para C, ya que el anterior no era válido.

Finalmente, en la línea 210 se resta del total, contenido en la variable D, la cantidad apostada. Las línea situadas a continuación corresponderán a la segunda parte del programa. Zona que se encargará de realizar la carrera propiamente dicha.

CONDICIONES PARA LA APUESTA:



IF A>0 AND A<= 1500 AND <= D THEN GO TO 180

En el programa se comprueba la validez de la apuesta examinando el contenido de la variable A. Las tres condiciones impuestas se funden en una sóla expresión por medio del operador lógico AND.

No va más...

La segunda zona del programa debe simular en la pantalla la carrera de los cinco veloces «purasangre». A modo de recordatorio, se indicará la cantidad y el número del caballo apostados; desde luego, después de haber utilizado el comando CLS para dejar limpia la «pista de carreras».

Acto seguido se trazará la línea de salida y la línea de llegada. Para ello se hace uso de la sentencia repetitiva FOR/NEXT, junto con los correspondientes PRINT AT, que imprimirá un signo consistente en una barra vertical (o el signo más parecido de que se disponga, para lo que habrá que consultar el manual del equipo utilizado y buscar en el repertorio de caracteres).

Queda aún una tarea previa, antes de pasar a codificar la zona que se ocupará de «abrir los cajones de los caballos»: inicializar las variables para las coordenadas horizontal y vertical de cada caballo. Estas coordenadas se almacenarán en cada uno de los cinco elementos de las matrices X y C. Como se recordará, ambas fueron dimensionadas en la línea 100 del segmento anterior.

En la variable suscrita X() se almacenará el valor actual de la coordenada horizontal. Su contenido debe ser incrementado (de forma aleatoria), puesto que la carrera transcurrirá desde el margen izquierdo hasta el derecho de la pantalla. La matriz C(), en cambio, mantendrá siempre valores constantes, que corresponden a la coordenada vertical; valores que delimitan la línea o «calle» por la que avanzará cada caballo.

El contenido inicial de esta matriz marca, por lo tanto, la línea imaginaria que seguirá cada caballo. Para lograr una distribución simétrica y evitar que «se molesten unos a otros», se elegirán de tal forma que queden tres líneas en blanco entre cada dos caballos. Así, los valores elegidos, siempre considerando una pantalla de 24 líneas horizontales, pueden ser: 3, 7, 11, 15 y 19.

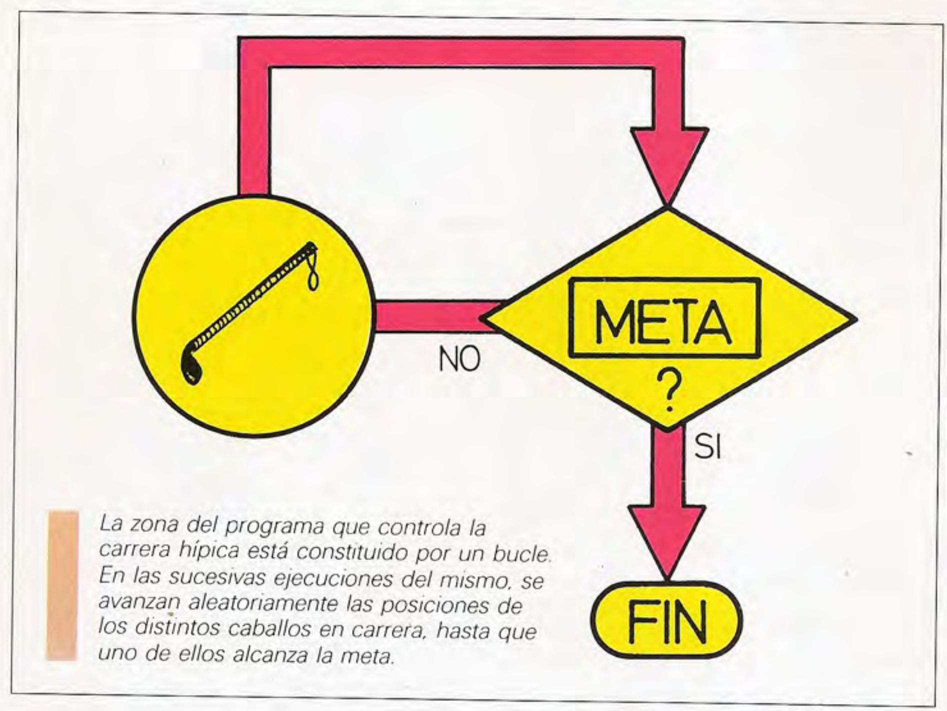
El hecho de almacenar estos valores en una variable suscrita se debe a que simplifica en gran medida la zona del programa que se encargará de situar a cada caballo en su lugar correspondiente. Así, se puede incluir esta acción en un bucle FOR/NEXT. De otra forma se necesitaría una línea de programa para cada uno de los cinco caballos. Ello haría quizás más rápida la ejecución del programa, pero, evidentemente, el listado sería más extenso. Este segmento del programa adopta el siguiente aspecto:

```
220 CLS
230 PRINT AT 4,0;"APOSTADAS ";A;" PTS. AL CABALLO ";C
240 FOR L=1 TO 21:PRINT AT 2,L;"|":PRINT AT 37,L;"|":NEXT L
250 LET C(1)=3:LET C(2)=7:LET C(3)=11:LET C(4)=15:LET C(5)=19
260 FOR I=1 TO 5:LET X(I)=0:LET H=C(I): GOSUB 500:NEXT I
```

Como ya se indicó, en dicha porción del programa se muestran la cantidad y el caballo que son objeto de la apuesta. Acción que se realiza en la línea 230. A continuación (línea 240) se trazan los límites de la pista por medio del bucle anteriormente explicado.

El objeto de las instrucciones contenidas en la línea 250 no es otro que dar valores a los elementos de C(); valores que coinciden con los propuestos más arriba.

La línea 260 utiliza de nuevo una instrucción iterativa FOR/NEXT. Esta se repite tantas veces como caballos corren (5). El mismo bucle se aprovecha para



La zona del programa que controla la carrera hípica está constituido por un bucle. En las sucesivas ejecuciones del mismo, se avanzan aleatoriamente las posiciones de los distintos caballos en carrera, hasta que uno de ellos alcanza la meta.

inicializar la matriz X con ceros (LET X(I)=0), que son las coordenadas horizontales de partida de los caballos. Dentro del bucle se llama a una subrutina que se situará a partir de la línea 500. La misión de esta subrutina es la de mostrar en pantalla cada caballo en su posición actual. Justo antes de invocar a la subrutina, se almacena en la variable H la coordenada vertical del caballo en curso (recuérdese que en cada pasada la variable I toma un valor diferente, este valor sirve para identificar a cada caballo).

La referida subrutina aparece codificada en las siguientes líneas:

```
500 LET M=X(1)*(H=C(1))+X(2)*(H=C(2))+ X(3)*(H=C(3))+X(4)*(H
    =C(4))+X(5)*(H=C(5))
510 PRINT AT M,H;"<";(H+1)/4;">"
520 RETURN
```

Su funcionamiento es el siguiente: mediante un comando PRINT AT se selecciona el punto de la pantalla en el que se va a situar al caballo. Para ello, la coordenada horizontal se calcula mediante la expresión de la línea 500:

```
M=X(1)*(H=C(1))+X(2)*(H=C(2))+X(3)*(H=C(3))+X(4)*(H=C(4))+
X(5)*(H=C(5))
```

Esta expresión se puede dividir para su estudio en dos partes: una lógica y otra aritmética. La parte lógica calcula el valor lógico (0 ó 1) de los términos del tipo H=C(j), siendo j un valor comprendido entre 1 y 5. Sólo una de las cinco expresiones será cierta: aquella cuyo valor actual de H coincida con el valor correspondiente de la matriz C(). Por lo tanto, ésta adoptará el valor lógico 1, mientras que las demás tomarán el valor 0. La zona aritmética se limita a multiplicar ese valor lógico por el valor actual de la coordenada horizontal.

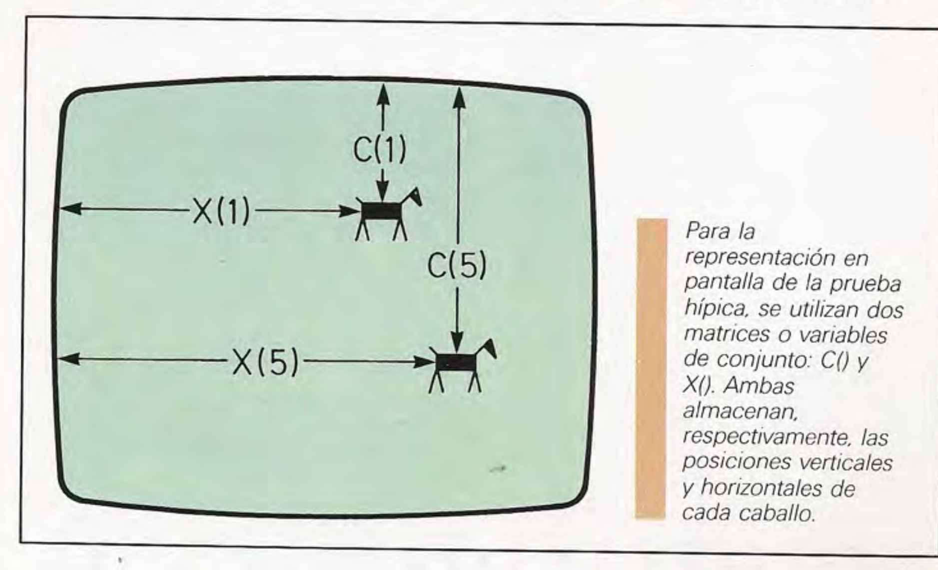
Esta última se encuentra almacenada en la matriz X(). Así, por ejemplo, si al efectuar la llamada a la subrutina se asigna a la variable auxiliar H el valor contenido en C(2), sólo el paréntesis que contiene a la expresión H=C(2) adoptará el valor 1. Todos los demás valdrán 0, con lo que el producto de ese 1 por X(2) dará como resultado el valor contenido en X(2), siendo nulos los demás sumandos.

La otra coordenada corresponde simplemente al valor de H. Una vez conocida la posición, se imprime en pantalla el símbolo del caballo. Para ello se hace uso del número de cada animal. Este se calcula a partir del valor de la variable H. La relación entre H (posición vertical del caballo en curso) y el número identificativo del caballo viene dada por la fórmula:

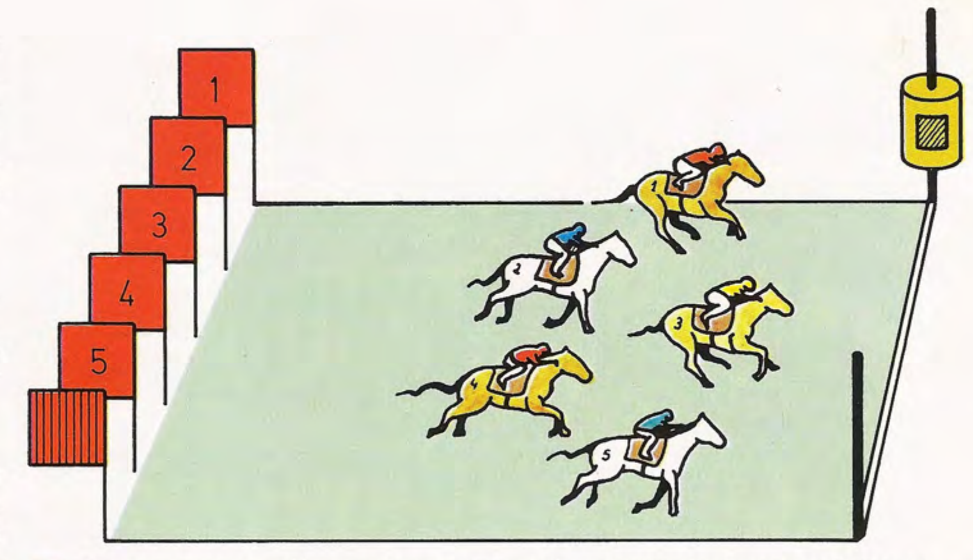
```
<número>=(H+1)/4
```

Esta fórmula habrá de ser corregida en el caso de elegir unas coordenadas verticales distintas a las aquí utilizadas.

En este ejemplo se ha sacrificado la estética en función de la estandarización. Por ese motivo los caballos se representarán simplemente por medio de su número, escoltado por los símbolos "<" y ">". Justo delante de estos tres símbolos ("<", número de caballo y ">") se imprimirá un espacio en blanco. Este tiene como misión borrar la estela que iría dejando cada caballo al avanzar por la pista, debido a la impresión sucesiva de los símbolos "<1>".



Para la representación en pantalla de la prueba hípica, se utilizan dos matrices o variables de conjunto: C() y X(). Ambas almacenan, respectivamente, las posiciones verticales y horizontales de cada caballo.



En el programa confeccionado se contempla la participación de cinco caballos en carrera; este número puede modificarse sencillamente sin más que introducir ligeras variaciones en el listado final.

Llegado el programa a este punto, tan sólo queda, para poner fin a esta segunda parte, la zona encargada de hacer correr a los caballos y de detectar la llegada del ganador a la meta. El listado de las líneas que corresponden a dicha zona es el que figura a continuación.

```
270 FOR J=1 TO 5
280 LET H=C(J):LET X(J)=X(J)+INT(RND+ 0.5):GOSUB 500
290 IF X(J)>33 THEN LET J=6:NEXT J:GOSUB 600:GOTO
    110
300 NEXT J:GOTO 270
```

Todo este segmento está incluido en un par de bucles. El primero de ellos (creado por el GOTO 270 de la línea 300), se encarga de repetir todo el bloque un número de veces suficientemente grande (teóricamente, infinitas veces). Con ello se logra tener la certeza de que alguno de los caballos llega a la meta. La llegada se comprueba en la línea 290, simplemente examinado si la coordenada horizontal de alguno de los caballos es superior a 33. Si es así, se dará por finalizado el bucle interno, poniendo su contador (J) al máximo de la cuenta.

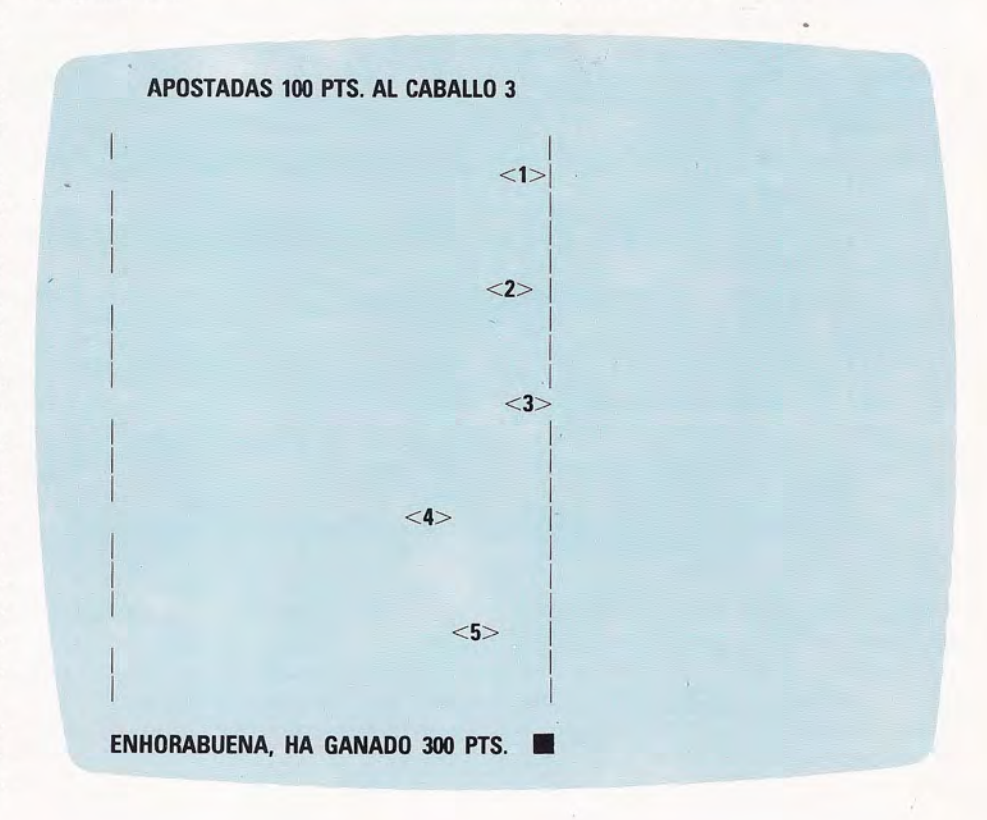
Seguidamente se verifica si el caballo ganador coincide con el apostado. Esto último se ha de realizar es la subrutina colocada a partir de la línea 600. Más tarde se regresa a la línea 110 (GOTO 110) para que tenga lugar una nueva apuesta.

El bucle más interno (formado por los comandos FOR y NEXT de las líneas 270 y 300) se repetirá una vez por cada caballo, siendo la línea 280 la que hará avanzar aleatoriamente a los equipos. Ello se efectúa mediante el comando RND, al que se añade la cantidad 0.5, de tal forma que el resultado de todo ello tendrá la misma probabilidad de ser 0 ó 1. Una explicación más a fondo de los usos de la función RND se encuentra en el siguiente capítulo de este mismo tomo.

El resultado de dicha operación es sumado al antiguo valor de la coordenadora horizontal. Una llamada a la subrutina de la línea 500 sitúa al caballo correspondiente en el nuevo punto de la pantalla.

La salida de este segmento de programa tiene lugar en la línea 290, mediante la llamada a una nueva subrutina situada a partir de la línea 600. Subrutina que constituye lo que hasta ahora se ha venido denominando «tercera parte del programa» y cuyo listado se ofrece a continuación:

```
600 IF C=(H+1)/4 THEN PRINT AT 3,23; "ENHORABUENA,
    HA GANADO ";A*3;" PTS.":LET D=D+A*3:GOTO 620
610 PRINT AT 3,23 "LO SIENTO, GANO EL CABA-
    LLO";(H+1)/4
620 FOR E=1 TO 1000:NEXT E:RETURN
```



```
10 REM TURF
20 CLS
30 PRINT AT 14,1 ; "*** TURF ***"
40 PRINT AT 0,6 ; "DISPONE INICIALMENTE DE 4.000 PTS."
50 PRINT : PRINT "LA APUESTA MAXIMA ES DE 1.500 PTS."
60 PRINT : PRINT "SI GANA EL CABALLO APOSTADO, SE"
70 PRINT "MULTIPLICARA POR TRES LA CANTIDAD JUGADA"
80 PRINT AT 2,23; "PULSE UNA TECLA PARA COMENZAR"
90 LET B$ = INKEY$ : IF B$ = "" THEN GOTO 90
100 DIM X(5):DIM C(5):LET D=4000
110 CLS:IF D=0 THEN PRINT "NO QUEDA DINERO PARA APOSTAR":END
120 PRINT "TIENE ";D;" PTS."
130 PRINT
140 INPUT "CUANTO APUESTA ";A
150 IF A>0 AND A<=1500 AND A<=D THEN GOTO 180
160 PRINT "APUESTA NO VALIDA"
170 GOTO 130
180 PRINT
190 INPUT "(DEL 1 AL 5) A QUE CABALLO ";C
200 IF C<1 OR C>5 THEN GOTO 180
210 LET D=D-A
220 CLS
230 PRINT AT 4,0;"APOSTADAS ";A;" PTS. AL CABALLO ";C
240 FOR L=1 TO 21:PRINT AT 2,L;"|":PRINT AT 37,L;"|":NEXT L
250 LET C(1)=3:LET C(2)=7:LET C(3)=11:LET C(4)=15:LET C(5)=19
260 FOR I=1 TO 5:LET X(I)=0:LET H=C(I):GOSUB 500:NEXT I
270 FOR J=1 TO 5
280 LET H=C(J):LET X(J)=X(J)+INT(RND+0.5):GOSUB 500
290 IF X(J)>33 THEN LET J=6:NEXT J:GOSUB 600:GOTO 110
300 NEXT J:GOTO 270
500 LET M=X(1)*(H=C(1))+X(2)*(H=C(2))+X(3)*(H=C(3))+X(4)*(H=C(4))+X(5)*
(H=C(5))
510 PRINT AT M,H;" <";(H+1)/4;">"
520 RETURN
600 IF C=(H+1)/4 THEN PRINT AT 3,23;"ENHORABUENA, HA GANADO ";A*3;" PTS.":LET
D=D+A*3:GOTO 620
610 PRINT AT 3,23 "LO SIENTO, GANO EL CABALLO ";(H+1)/4
620 FOR E=1 TO1000:NEXT E:RETURN
```

Esta última parte no tiene mayor complicación que la de comprobar si el caballo ganador coincide con el que recibió la apuesta. De darse esta circunstancia, se procederá a calcular el importe ganado, mediante una simple multiplicación por tres. En el caso de que el pronóstico no haya sido el acertado, se imprimirá el correspondiente mensaje.

Antes de devolver el control al punto de llamada de la subrutina, se establece un retardo mediante un nuevo bucle FOR/NEXT vacío. Retardo que permite ver el resultado. Una vez transcurrido ese tiempo se regresará al punto de partida.

Desde allí se salta a la línea 110, donde dará comienzo una nueva apuesta. El programa ha sido ya elaborado en su totalidad y ofrece el aspecto que refleja el listado adjunto.

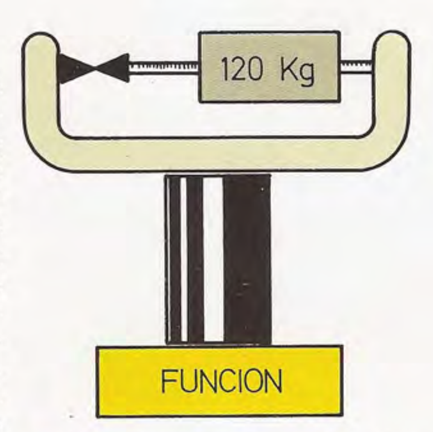
Funciones BASIC

Introducción de datos, funciones numéricas y de azar



La mayor parte de las palabras BASIC mencionadas hasta ahora son comandos. En el primer capítulo dedicado el manejo de cadenas alfanuméricas se introdujeron palabras que no eran comandos, sino funciones. La diferencia entre ambos tipos de órdenes estriba en su funcionamiento: si los comandos realizan acciones, las funciones ejecutan operaciones. Ello significa que el efecto de una función será emitir un resultado en forma de dato.

El uso del comando PRINT dentro de una instrucción, por ejemplo, PRINT "HOLA", produce una acción; no emite un dato resultante de una operación. Dicha acción corresponde con la presentación en pantalla de la palabra HOLA. Sin embargo, la función LEFT$ (1, "HOLA") ejecuta una opeación cuyo resultado es el dato "H". Así pues, función es toda herramienta del vocabulario BASIC que proporciona un dato. A lo largo del presente capítulo se van a estudiar algunas funciones de interés, que servirán para sacar un mayor partido a la máquina.





La diferencia entre una función y un comando reside en el hecho de que la función proporciona un dato de salida, mientras que el comando realiza una función.

Introducción de datos

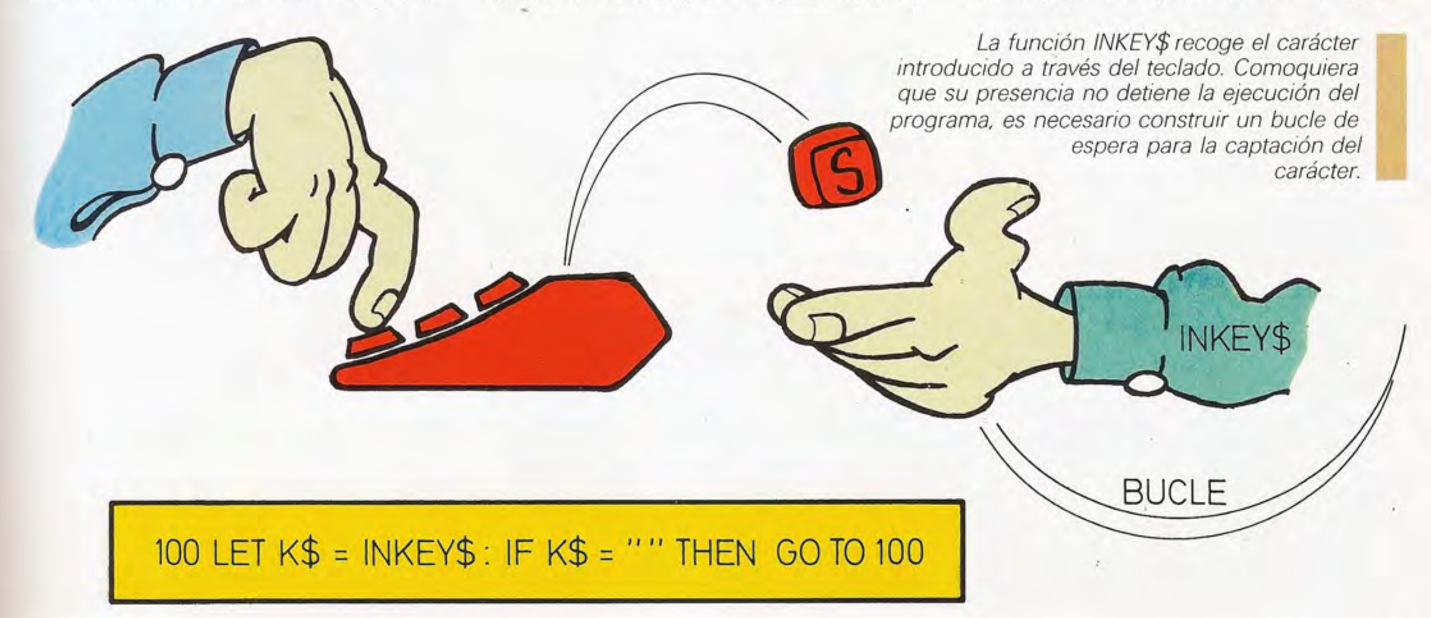
Cuando en un programa es necesario introducir nuevos datos en cada ejecución, ello debe indicarse dentro del propio programa. Tal cometido se realizaba hasta ahora por medio del comando INPUT. A pesar de su indudable utilidad, éste comando presenta ciertos inconvenientes. Cuando la cantidad de datos a introducir es grande y se exige cierta rapidez, la actuación de INPUT deja bastante que desear, puesto que obliga a pulsar la tecla RETURN tras cada dato introducido. El BASIC incorpora una función especializada en la introducción de datos. Esta es INKEY$, la cual proporciona el carácter pulsado en el teclado. La diferencia entre INPUT e INKEY$ se observará con claridad a través del siguiente ejemplo.

A menudo, se utilizan decisiones aportadas por el usuario para desviar la ejecución del programa. Apelando al comando INPUT, la forma de resolver tal necesidad puede coincidir con la que refleja el siguiente bloque de instrucciones:

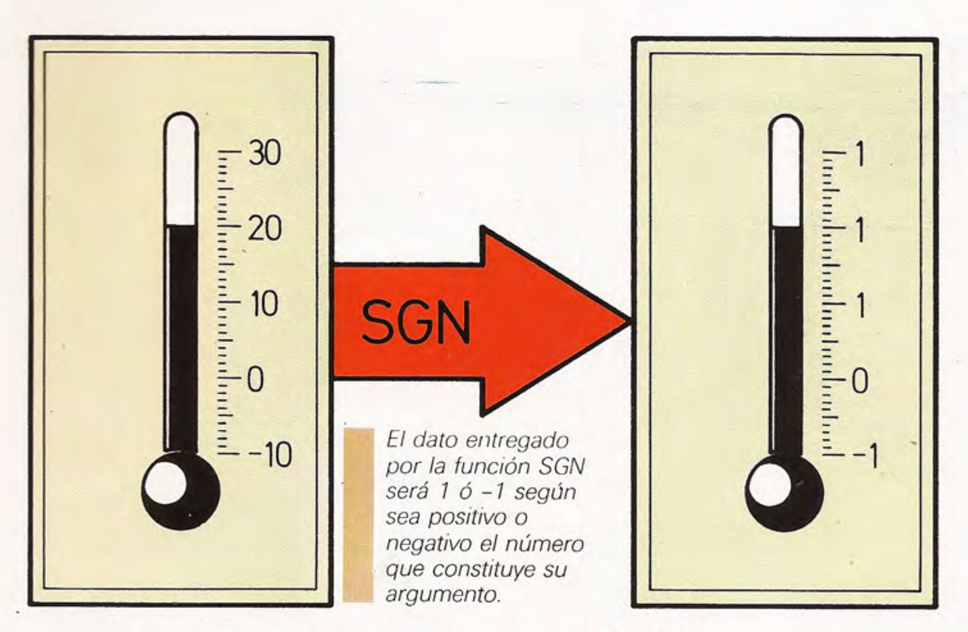
```
...
200 PRINT "DESEA CONTINUAR (S/N)?"
210 INPUT K$
220 IF K$="S" THEN GOTO...
230 IF K$="N" THEN GOTO...
...
```

Al llegar a esta zona del programa, el operador ha de teclear un dato (S ó N) y luego pulsar RETURN.

Si opta por utilizar la función INKEY$, ya no será necesario accionar la tecla RETURN, puesto que dicha función recoge un solo carácter del teclado. Sin embargo, INKEY$ no detiene la ejecu-



La función INKEY$ recoge el carácter introducido a través del teclado. Comoquiera que su presencia no detiene la ejecución del programa, es necesario construir un bucle de espera para la captación del carácter.



El dato entregado por la función SGN será 1 ó −1 según sea positivo o negativo el número que constituye su argumento.

ción, con lo cual, si no se pulsa la tecla en el momento oportuno, no se obtendrá ningún dato. En consecuencia, no será suficiente con cambiar la línea 210 por:

```
210 K$=INKEY$
```

Para evitar que la ejecución «pase de largo» es necesario crear un bucle; bucle que ha de permanecer cerrado sobre sí mismo hasta que se detecte la pulsación. Las instrucciones necesarias para programarlo se pueden agrupar en una misma línea:

```
210 K$=INKEY$: IF K$="" THEN GOTO 210
```

Una vez que esta línea entre en actividad, la secuencia de ejecución permanecerá virtualmente bloqueada en la misma hasta que se accione una tecla.

En definitiva, el nuevo aspecto del ejemplo resuleto con la función INKEY$ será el que sigue:

```
...
200 PRINT "DESEA CONTINUAR (S/N)?"
210 K$=INKEY$: IF K$="" THEN GOTO 210
220 IF K$="S" THEN GOTO...
230 IF K$="N" THEN GOTO...
...
```

En algunos ordenadores no existe la función INKEY$. Cuando esto ocurre, suele utilziarse en su lugar un nuevo comando: GET. Lo mismo que antes se conseguía con LET K$=INKEY$, se obtiene ahora con GET K$. Aplicando el nuevo comando al ejemplo anterior, sólo habría que cambiar una línea:

```
210 GET K$: IF K$="" THEN GOTO 210
```

Jugando con números

Suponga un programa en el que se desea realizar dos acciones distintas, dependiendo de si un número es par o impar. La forma de resolverlo es haciendo uso de la instrucción IF/THEN. Pero, ¿cómo detectar si el número es par?

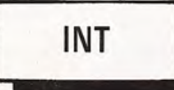
Sabemos que un número es par si es divisible por dos. Así, por ejemplo: 6/2=3 y 5/2=2,5; el número 6 es par mientras que el 5 es impar. La diferencia radica en la parte fraccionaria, de tal forma que un número es par si el resultado de dividirlo por dos no posee parte fraccionaria.

Para separar la parte entera de la parte fraccionaria se dispone, en BASIC, de la función INT. Esta devuelve la parte entera (INTeger, entero) del número situado en su argumento:

```
PRINT INT(16.123)
16
■
```

La función INT también permite calcular la parte fraccionaria. La parte fraccionaria de un número N será N-INT(N); esto es: el mismo número aunque desprovisto de la parte entera. A continuación se evalúa la parte fraccionaria del número del ejemplo anterior:

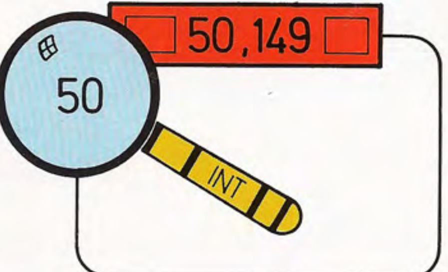
```
PRINT 16.123-INT(16.123)
0.123
■
```



Devuelve la parte entera de un número.

Formato: INT(<número>)

Ejemplos: 10 A=INT(17.5)
50 PRINT INT(B+C)



El cometido de la función INT es extraer la parte entera del dato numérico que se indique en el argumento.

Volviendo al problema del principio, se utilizará la función INT para detectar cuándo un número es par. Resulta obvio que la parte entera de un número entero coincidirá con el mismo número. Luego si N es entero INT(N) y N tendrán el mismo valor. Esto es, precisamente, lo que se va a utilizar para decidir si el dato NUM es par.

```
IF NUM/2=INT(NUM/2) THEN PRINT "PAR"
```

De esta forma, si NUM/2 es igual INT(NUM/2) entonces es que NUM/2 es entero. Y si NUM/2 es entero, resultará que NUM es par.

El mismo procedimiento se puede utilizar para averiguar si NUM es divisible por cualquier otro número. Sencillamente, cambiando el 2 por el número deseado. NUM/7=INT(NUM/7) será cierto si NUM es divisible por siete.

Mas funciones: cuestión de signos

Hay otras funciones que proporcionan resultados interesantes. Una de ellas es ABS, cuyo cometido es calcular el valor absoluto de un número. El siguiente ejemplo es una muestra de las posibilidades de ABS:

```
PRINT ABS(17)
17
PRINT ABS(—17)
17
█
```

La función ABS resulta especialmente útil para determinar la separación entre dos números (sean A y B). Si se realiza la resta A-B, el resultado tomará un valor positivo si A es mayor que B; en caso contrario, el resultado será negativo. Para obtener siempre un resultado positivo basta con poner ABS(A-B).

Una función íntimamente relacionada con ABS es SGN. Como su nombre indica, SGN identifica el «SiGNo» de su argumento. Entrega el valor 1 si el dato es positivo, -1 si es negativo y 0 si el dato que figura en su argumento es nulo.

Una aplicación de SGN puede ser la de adecuar el incremento de un bucle.

```
FOR I=A TO B STEP SGN(B—A)
```

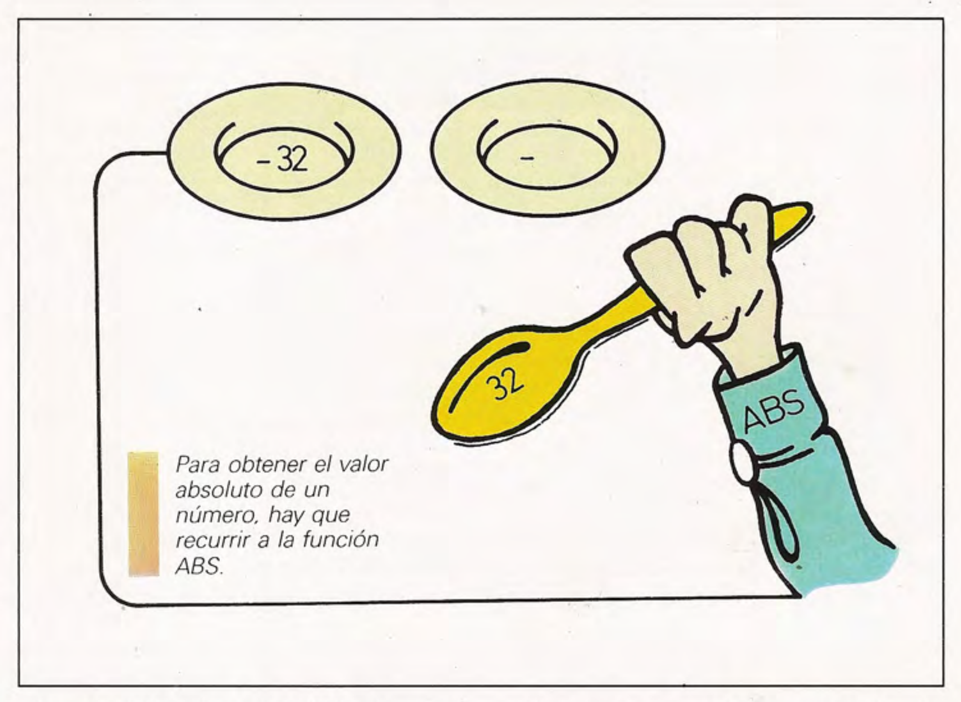
Se supone que el incremento ha de ser de una unidad y los valores de A y B no son conocidos. En consecuencia, si B es mayor que A el incremento ha de ser positivo, mientras que si es A el mayor, el incremento debe ser negativo. Ello se consigue con la cláusula STEP SGN(B-A). Cabe comprobar que cuando B es mayor que A, SGN(B-A) vale 1, y

ABS

Calcula el valor absoluto de un número.

Formato: ABS (<número >)

Ejemplos: 100 MODU=ABS(D)
120 PRINT ABS(Z)



Para obtener el valor absoluto de un número, hay que recurrir a la función ABS.

INKEY$

Recoge un carácter pulsado en el teclado.

Formato: <var.>=INKEY$

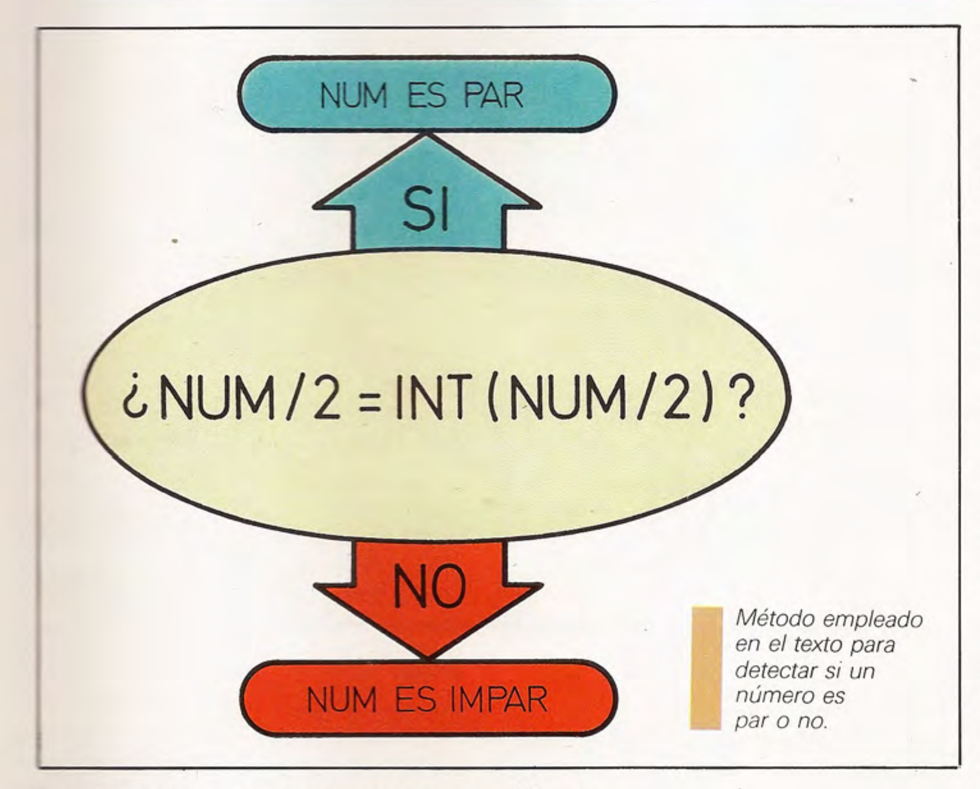
Ejemplos: 70 K$=INKEY$
50 B$=: KEY$

SGN

Obtiene 1 si el argumento es positivo, –1 si es negativo y 0 si es nulo.

Formato: SGN(<número>)

Ejemplos:
```
50 A=A+SGN(B)
70 PRINT SGN(N)
```



Método empleado en el texto para detectar si un número es par o no.

en caso contrario –1. Justamente lo que se pretendía conseguir. Si se desea un incremento de valor distinto de 1, e igual al valor INC, será suficiente con cambiar la parte STEP de la siguiente forma:

```
FOR I=A TO B STEP INC*SGN(B—A)
```

El azar

En muchos juegos y en ciertos programas de simulación, el «azar» tiene una gran importancia. En el juego del parchís es fundamental el uso de un dado. En una simulación ha de contemplarse la variación aleatoria de algún factor (temperatura, viento...). Esta componente aleatoria se puede simular perfectamente en el ordenador.

El problema consiste en obtener una secuencia de números que no sigan (aparentemente) una regla fija. A cada uno de dichos números se le puede asignar, posteriormente, la cara de una moneda o de un dado, etc. De esta forma será posible evaluar los resultados de las distintas tiradas.

Una posible forma de calcular una secuencia «pseudoaleatoria» puede ser la siguiente:

1. Se parte de un número del 1 al 10.
2. Se eleva al cuadrado
3. Se toma, del cuadrado, el dígito de mayor peso.
4. Se vuelve al paso 2.

Aplicando este método se obtendrán números de una cifra, sin una aparente relación entre sí. Véase, por ejemplo, la secuencia resultante a partir del número 9:

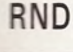
9 (9×9=81—8)
8 (8×8=64—6)
6 (6×6=36—3)
3 (3×3=9—9)
9 ...

La sencillez de este método trae consigo varios problemas. En el ejemplo se ve que a partir del cuarto número la secuencia se repite. El caso peor es cuando se parte del valor 1. Al elevarlo al cuadrado se obtiene de nuevo un 1 y, por lo tanto, la secuencia se repite con el mismo valor.

Existen métodos más complejos para el cálculo de secuencias «pseudoaleatorias». Muchos de ellos son fruto de la aplicación de teorías estadísticas y de probabilidad.

En la amplia mayoría de los dialectos BASIC, existe una función adecuada para calcular números pseudoaleatorios. A esta función se accede por medio de la palabra clave RND (de RaNDom, aleatorio en inglés). Generalmente RND precisa de un dato de partida o «semilla» de la secuencia pseudoaleatoria.

En algunos aparatos, la función RND sólo calcula el siguiente número de la secuencia, partiendo de la semilla. En



RND

Genera un número aleatorio comprendido entre cero y uno.

Formato: RND[(<número>)]

Ejemplos:
```
A=RND(1)
B=RND
```

TABLA DE CONVERSION							
Ordenador	RND	RANDOMIZE	INKEY$	GET	INT	ABS	SGN
	RND [(<c>)]	RANDOMIZE	INKEY$	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
AMSTRAD	RND (<c>)	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
APPLE II (APPLESOFT)	RND (<c>)	RND(-<c>)	—	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
APRICOT (M-BASIC)	RND [(<c>)]	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
ATARI	RND (<c>)	—	—	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
CBM 64	RND (<c>)	RND (<c>)	—	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
DRAGON	RND (<A>)(1)	—	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
EQUIPOS MSX	RND (<c>)	RND (-<c>)	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
HP-150	RND [(<c>)]	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
IBM PC	RND [(<c>)]	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
MPF	RND (<c>)	—	—	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>
NCR DM-V (MS-BASIC)	RND [(<c>)]	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
NEW BRAIN	RND	RANDOMIZE	—	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
ORIC	RND	RND (-<c>)	KEY$	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
SHARP MZ-700 (MZ-BASIC)	RND (1)	RND (-<c>)	—	GET <variable>	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
SINCLAIR QL	RND [<A>[TO<B>]](2)	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
SPECTRAVIDEO	RND (<c>)	RND (-<c>)	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)
ZX-SPECTRUM	RND	RANDOMIZE	INKEY$	—	INT (<núm.>)	ABS (<núm.>)	SGN (<núm.>)

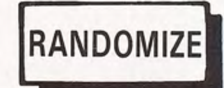
<c>: Indica un número positivo cualquiera. No importa el valor de dicho número. <variable>: Nombre de variable de cadena alfanumérica. <núm.>: Constante, variable o expresión numérica.

(1) Calcula un número entero entre 0 y A. Si A vale 0 calcula un número entre 0 y 1.

(2) Genera un entero aleatorio entre A y B. Si sólo se especifica A, genera un entero entre 0 y A.

ellos la semilla debe especificarse entre paréntesis, a continuación de RND. De esta forma, siempre que se ejecute RND (<número>) y el <número> sea el mismo, se obtendrá idéntico resultado. Para encadenar los resultados y que el número calculado sirva de semilla al siguiente, se puede hacer algo semejante a lo que sigue:

```
10 INPUT SEM
20 SEM=RND(SEM)
```



Introduce una «semilla» para generar una nueva secuencia aleatoria.

Formato: RANDOMIZE[<semilla>]

Ejemplos:
```
100 RANDOMIZE
50 RANDOMIZE 7
```

```
30 PRINT SEM
40 GOTO 20
```

Con ello se consigue una secuencia en la que cada nuevo número sirve de semilla al siguiente. En un programa complejo se utilizaría SEM como número aleatorio y se volvería a realizar el cálculo de forma análoga a la línea 20.

En ciertos dialectos BASIC la función RND no utiliza argumento, o carece de importancia el argumento que se le proporcione. En esos casos, dicha función se encarga por sí misma de guardar y reutilizar el último número generado. Para inicializar la semilla se dispone, entonces, de un comando adjunto: RANDOMIZE, en cuyo argumento se indica la semilla que ha de iniciar la secuencia.

Sea de una o de otra forma, la función RND dará, por lo general, un número decimal comprendido entre 0 y 1, pero que nunca llegará a alcanzar 1 (será un dato del tipo 0, XYZ).

Utilización del azar

Haciendo uso de la función RND se puede simular cualquier fenómeno de tipo aleatorio; por ejemplo, el lanzamiento de un dado. Veamos cómo es posible construir un programa BASIC que calcule la puntuación asociada a la tirada de un dado.

El corazón del programa será, en efecto, la función RND. El único problema reside en que dicha función calcula un número comprendido entre 0 y 1. Así pues, lo primero que hay que hacer es convertir dicho número en un entero menor que siete y mayor que cero. El primer paso consiste en multiplicar el número aleatorio por 6.

De esta forma, los posibles datos estarán comprendidos entre 0 y 6; el número resultante podrá ser: cero con algo, uno con algo, ..., hasta cinco con algo. Está claro que lo que interesa es la parte entera de este resultado. El segundo paso será, pues, hallar la parte entera mediante la función INT.

Al aplicar INT se obtiene un número entero comprendido entre el cero y el cinco. Ahora sólo falta sumar una unidad para obtener un número entero entre 0+1=1 y 5+1=6. Este será el aspecto del programa BASIC.

```
10 REM DADO
20 INPUT "SEMILLA";S
30 RANDOMIZE S
40 PRINT 1+INT(6*RND)
50 GOTO 40
```

La línea 40 es la que realiza los cálculos explicados más arriba. En general, para obtener un número entero comprendido entre N (mínimo) y M (máximo), ambos inclusive, la fórmula a aplicar es la siguiente:

```
DATO=N+INT((M=N+1)*RND)
```

Si se tiene en cuenta que RND proporciona números «equiprobables» (esto es: en cualquier momento todos los números posibles tienen la misma probabilidad de aparecer), se puede construir un «lanzador de monedas» con la simple instrucción:

```
IF RND>0.5 THEN PRINT "CARA" ELSE PRINT "CRUZ"
```

Ello también se puede conseguir trasladando el rango de los resultados de RND, como en el caso del dado.

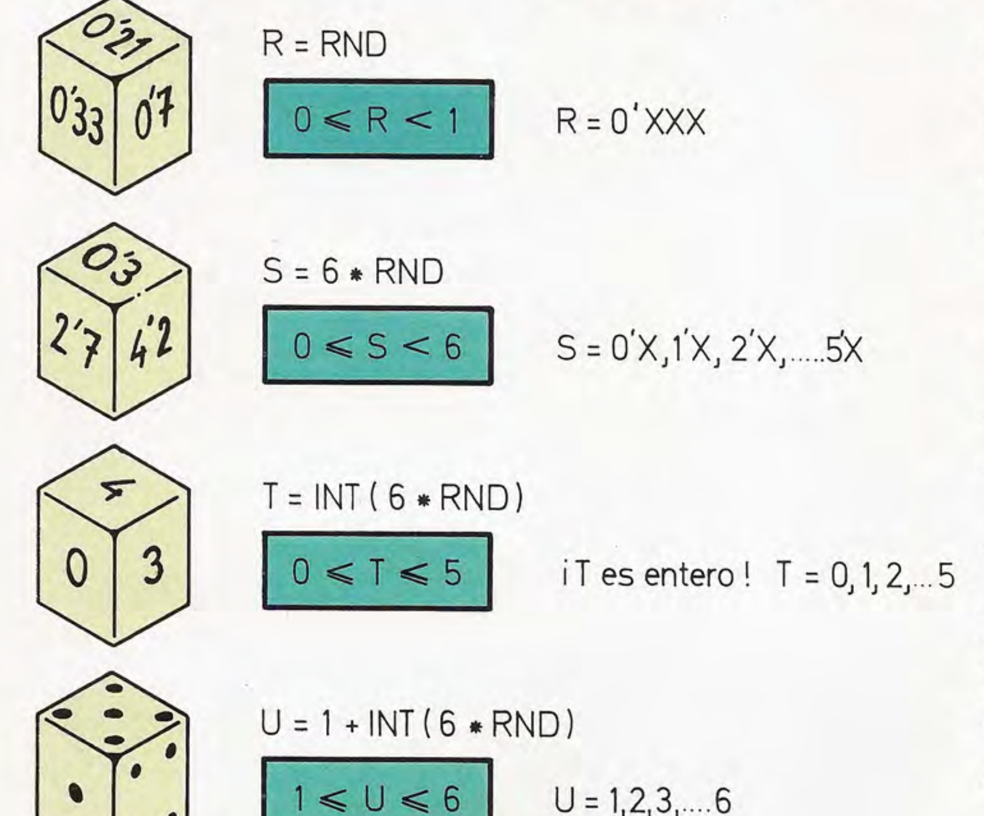
Otro método alternativo consiste en la puesta en práctica de la función SGN.

```
MONE=SGN(RND—0.5)
```

Con este último ejemplo se obtienen los valores 1 ó −1 con igual probabilidad. El contenido de MONE puede ser utilizado posteriormente para tomar una decisión aleatoria. En el siguiente ejemplo se consigue un aumento o disminución, de forma aleatoria del contenido de DATO.

```
DATO=DATO+SGN(RND—0.5)
```

En algunos intérpretes BASIC, la función RND da directamente un número entero. En tal caso, el rango de posibilidades se especifica en el argumento de RND.



Secuencia de operaciones a realizar para modificar el rango de los números generados a partir de la función RND, en orden a simular el lanzamiento de un dado.

Funciones matemáticas

Las herramientas de cálculo del lenguaje BASIC

Los ordenadores nacieron como una herramienta de cálculo. La historia de la informática se remonta a las antiguas máquinas de calcular mecánicas. Quizás el más antiguo predecesor de los modernos ordenadores sea el ábaco, con el que se podían realizar sumas, restas y hasta multiplicaciones.

Los primeros ordenadores electrónicos no iban mucho más allá. Hoy en día la actividad del ordenador abarca prácticamente todos los campos, pero sin duda, el cálculo matemático es uno de los principales.

Todos los lenguajes de programación incluyen, entre su repertorio de instrucciones, funciones de tipo matemático. El BASIC es un lenguaje derivado de otro más antiguo: el FORTRAN. Este último estaba claramente orientado a los problemas de índole científica y matemática, y por ello aportaba funciones matemáticas de fácil uso. El BASIC ha heredado del FORTRAN esta facilidad para el tratamiento de datos numéricos.

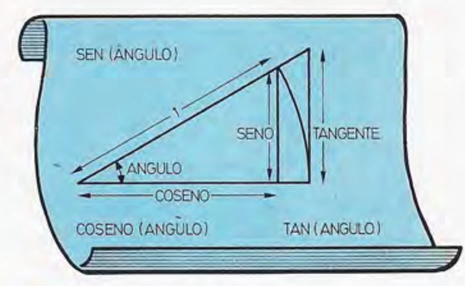
En capítulos anteriores se ha hablado de los operadores matemáticos elementales. Con ellos se podían realizar operaciones tales como suma (+), resta (–), multiplicación (*), división (/) y potenciación (ˆó↑). Se vio también su uso en sentencias de asignación y como parámetros calculados en el argumento de un comando. Los siguientes ejemplos muestran estas posibilidades.

```
LET S=Aˆ2+5*B
PRINT Aˆ2+5*B
```

La capacidad de cálculo en BASIC no queda ahí. Este lenguaje está capacitado para realizar operaciones más complejas.



Las posibilidades de cálculo de las funciones matemáticas del BASIC no distan mucho de las de una calculadora científica.



La resolución de cálculos trigonométricos es una de las vertientes contempladas en BASIC. Las funciones básicas para el cálculo del seno, coseno y tangente están en casi cualquier dialecto BASIC.

Raíces cuadradas

El lenguaje BASIC dispone de una función específica para el cálculo de raíces cuadradas. Esta función ha sido utilizada con anterioridad dentro de la obra, para realizar algunos pequeños cálculos.

Su formulación es muy sencilla, se apoya en la palabra clave SQR (del inglés SQuare Root, raíz cuadrada). Dicha palabra clave ha de ir seguida por el argumento, que no es más que el dato del que se desea extraer la raíz, situado entre paréntesis. Así pues, para hallar la raíz cuadrada de 2 basta con escribir SQR(2).

Esta, al igual que las restantes funciones de tipo matemático, debe estar acompañada por un comando BASIC; por ejemplo, puede ir precedida por el comando LET dentro de una sentencia de asignación:

```
10 LET RAIZ=SQR(2)
```

El argumento de SQR ha de contener un dato numérico entre paréntesis. Este dato puede pertenecer a cualquiera de los títulos normales en BASIC: constante, variable o expresión. Así, se puede indicar el cálculo de la raíz cuadrada de una variable, con lo que se consigue una mayor flexibilidad de uso.

Las expresiones matemáticas introducidas en el argumento pueden incluir, normalmente, cualquier operador o función matemática BASIC (incluso otra raíz cuadrada); por ejemplo:

```
LET RAIZ=SQR(A)
LET SOLU=SQR(6*A+1/2)
PRINT SQR(2+SQR(A/B))
```

Otras funciones

La utilización de la función SQR es casi imprescindible para la resolución de cualquier problema matemático. Al profundizar algo más en el cálculo matemático se hace necesario el uso de funciones más complejas. El siguiente paso son las funciones trigonométricas. En BASIC se puede hacer uso del seno, coseno y tangente mediante las palabras claves SIN, COS y TAN respectivamente. Al igual que la función SQR, éstas llevarán su dato de entrada entre paréntesis. El dato puede ser de cualquiera de los tres tipos anteriormente enumerados.

```
LET A=SQR(SIN(X)+COS(X))
PRINT TAN(A+2*B)
```

Estos dos ejemplos ilustran las posibilidades de las funciones trigonométricas. El ángulo sobre el que actúan se debe dar, por regla general, en radianes. De esta forma, para calcular el seno (coseno o tangente) de un ángulo expresado en grados, hay que pasar dicho ángulo previamente a radianes. En todo caso, se puede escribir un pequeño programa que realice tal conversión. Lo único que hace falta saber es que 180 grados corresponden a (número PI) radianes. Con este dato y mediante una sencilla regla de tres, se consigue la fórmula que permite pasar de grados a radianes: radianes = grados $*(\pi/180)$.

```
10 REM PASO DE GRADOS A RADIANES
20 LET PI=3.1416
30 INPUT "GRADOS";G
40 PRINT G*(PI/180);"RADIANES"
50 END
█
```

En este programa se incluye la fórmula de conversión directamente en el comando PRINT de la línea 40. Es importante introducir el valor de π directamente. En tal caso se puede acceder a su valor poniendo sencillamente el símbolo (π) o, en otros equipos, el nombre (PI). Con esta facilidad el programa se reduciría a la expresión siguiente:

```
10 REM PASO DE GRADOS A RADIANES
30 INPUT "GRADOS";G
40 PRINT G*( /180);"RADIANES"
50 END
█
```



El lenguaje BASIC ha heredado del FORTRAN su facilidad para el tratamiento de datos numéricos.

Tanto de una como de otra forma el programa proporciona el valor en radianes apto para ser incluido en el argumento de cualquier función trigonométrica. Su ejecución mostrará por pantalla un aspecto similar al siguiente:

```
RUN
GRADOS?45<CR>
,7853981675 RADIANES
█
```

De esta forma se puede hacer que el ordenador calcule senos, cosenos o tangentes de un ángulo dado en grados.

Esta misma filosofía permite construir funciones que trabajen directamente con grados. El siguiente ejemplo muestra cómo realizar un programa que calcula el seno de un ángulo cuyo valor se introduce en grados. Para el coseno o la tangente la actuación sería similar.

```
10 INPUT "SENO ";G
20 PRINT SIN(G* /180)
```

La expresión encerrada entre paréntesis es la que transforma los grados en radianes. Algunos ordenadores permiten elegir el modo de operación (grados o radianes). Para ello disponen de dos comandos: DEG y RAD. El primero hace que los ángulos se tomen en grados. Así, las funciones operarán en lo sucesivo con su argumento en grados. El último ejemplo se simplificaría adoptando ahora el siguiente aspecto:

```
 5 DEG
10 INPUT "SENO ";G
20 PRINT SIN(G)
```

El comando RAD restituye el modo de operación a radianes.

El resto de las funciones trigonométricas (secante, cosecante y cotangente) pueden ser calculadas a partir de estas tres. Su equivalencia es la siguiente:

```
Secante (X)=1/COS(X)
Cosecante (X)=1/SIN(X)
Cotangente (X)=1/TAN(X)
```

Cuando se utilizan las funciones trigonométricas elementales se hace necesario disponer de sus inversas (arcoseno, arcocoseno y arcotangente).

SIN

Calcula el seno natural del ángulo incluido en su argumento.

Formato: SIN (<ángulo>)

Ejemplos:
```
20 LET SENO=SIN(ANG)
70 PRINT SIN (2*X+1)
```

COS

Calcula el valor del coseno natural del ángulo incluido con su argumento.

Formato: COS(<ángulo>)

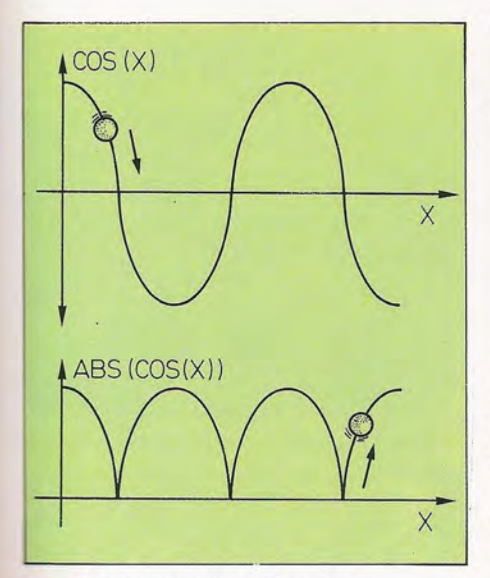
Ejemplos:
```
20 LET CON=COS(ANG)
70 PRINT COS(B+90)
```

TAN

Calcula el valor de la tangente del ángulo incluido en su argumento.

Formato: TAN(<ángulo>)

Ejemplos:
```
20 LET TGN=TAN(ANG)
70 PRINT TAN(1-45)
```



Representación gráfica de las funciones COS(x) y ABS (COS(x)). Por medio de ellas es posible simular la trayectoria de una pelota botando.

Estas son las que realizan la acción contraria a las anteriores. Así, si X=seno (A), sucede que A=arcoseno (X).

En BASIC estas funciones se formulan de la siguiente forma:: ASN (arcoseno), ACS (arcocoseno) y ATN (arcotangente). Su utilización es idéntica a la de SIN, COS o TAN.

```
50 LET A=ASN(0,5)
10 PRINT ACS(X/SQR(2))
```

En algunos aparatos tan sólo se dispone de una de las funciones inversas, por lo general ATN. Ello no plantea ningún problema ya que las otras dos son calculables a partir de ésta. Las fórmulas que relacionan estas funciones entre sí son:

```
ASN (X)=ATN (X/SQR(1-X^2))
ACS (X)=ATN (SQR(1-X^2)/X)
```

La deducción de estas fórmulas es sumamente fácil con unos ligeros conocimientos de trogonometría.

Logaritmos

Tan importante como el uso de las funciones trigonométricas es el de los logaritmos.

La mayoría de los intérpretes BASIC poseen una función capaz de calcular logaritmos. Esta función suele ser LOG, que da el valor del logaritmo neperiano o natural del dato incluido en su argumento. Este logaritmo es el que tiene como base el conocido número **e** (2,718281...).

Los siguientes son ejemplos válidos de uso de la función LOG

```
PRINT LOG(T+14)
LET X=5*LOG (17)
```

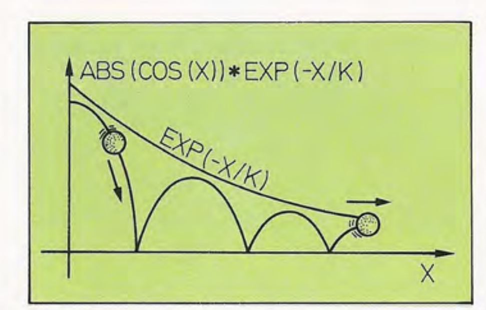
Partiendo del concepto de logaritmo:

$\mathrm{Log}_b x=y$

en donde se cumple que:

$x=b^y$

se observa que la función inversa resulta muy fácil de obtener. En las expresiones precedentes, b es la base (en este caso el número e). De ahí que para calcular el valor de x, conocido el valor de su logaritmo (y), será suficiente con elevar el número "e" (la base) a la potencia indicada (y). En definitiva, para hallar, en BASIC, el antilogaritmo de un número Y servirían las siguientes líneas.



Para que sea posible contemplar la amortiguación de los rebotes de la pelota, habrá que multiplicar la función ABS (COS(X)) por la expresión exponencial EXP (-X/K).

```
10 LET E=2.71828182
20 LET ANTILY=E^Y
```

No obstante, la mayor parte de los intérpretes BASIC poseen una función adecuada para su cálculo directo. Se trata de la función EXP. Así pues, EXP(Y) equivale a 2.71828182^Y. De esta forma, el anterior ejemplo quedaría reducido a:

```
20 LET ANTILY=EXP(Y)
```

Otros ordenadores disponen también de una función adecuada para hallar logaritmos cuya base es el número **10.** Dicha función es accesible por medio de la palabra clave LOG10.

El hecho de que sean pocos los ordenadores que incluyen esta última función no plantea inconveniente algino. Es perfectamente posible calcular logaritmos de cualquier base partiendo de los neperianos. Para deducir la fórmula que permite expresar logaritmos de una base cualquiera b mediante logaritmos neperianos, basta con conocer una simple propiedad de los logaritmos:

$K.\mathrm{Log}\ x=\mathrm{Log}\ x^k$

Así pues, si y es el logaritmo decimal de un número x, resulta cierto que:

$y=\mathrm{Log}_{10}x \Rightarrow x=10^y$

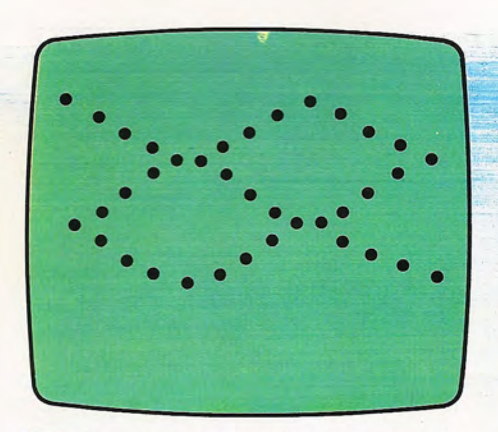
Si se aplican logaritmos a los dos miembros de la segunda expresión (Log_e= =Ln) queda:

$\mathrm{Ln}\ x=\mathrm{Ln}\ 10^y$

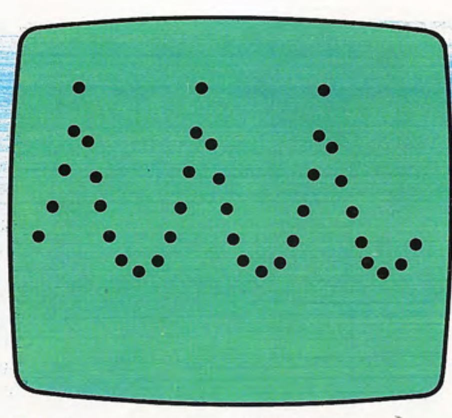
Según la propiedad anteriormente mencionada, se puede sustituir el segundo mimbro de la igualdad.

$\mathrm{Ln}\ x=y*\mathrm{Ln}\ 10$

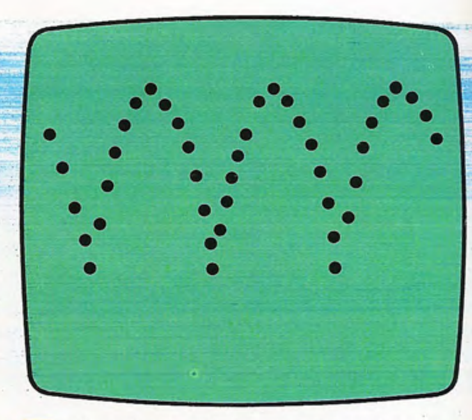
Ahora sólo queda despejar la variable



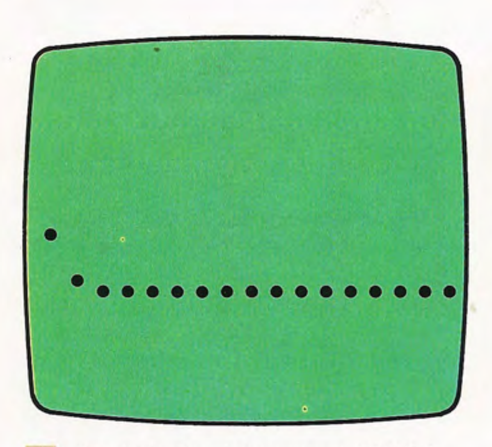
Aspecto de la pantalla al ejecutar el programa BOLA1: no llega a apreciarse con claridad el bote.



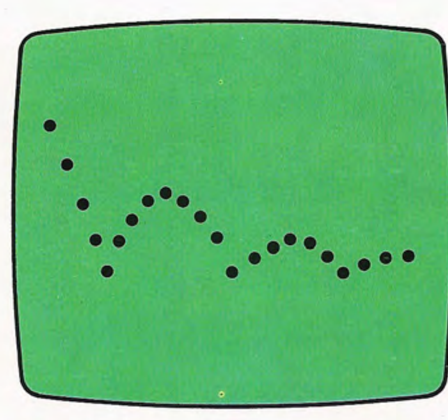
Con el programa BOLA2 es posible ya observar la trayectoria, aunque su trazado aparece invertido.



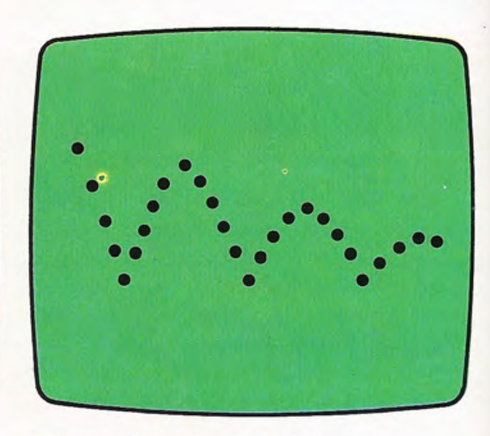
Simulación de la trayectoria de una pelota creada por la ejecución del programa BOLA3.



La ejecución del programa BOLA4 contempla la amortiguación de los sucesivos rebotes. En el caso ilustrado se otorga el valor 1 a la constante K: la amortiguación es instantánea y no da pie a rebote.



Si la constante K fuera igual a 15, la amortiguación sería ya más realista



e incluso puede realzarse asignando un valor superior a K, por ejemplo K = 30.

y, que coincide con el valor del logaritmo en base 10 de x. Con ello se obtiene la fórmula buscada:

$$y=\mathrm{Log}_{10}\ \frac{\mathrm{Ln}\ x}{\mathrm{Ln}\ 10}$$

Si lo que desea es calcular logaritmos de otra base, basta pues con indicar la nueva base en el argumento del logaritmo neperiano situado en el denominador de la expresión. Esta fórmula se codificaría en BASIC como sigue:

```
LET LBX=LOG(X)/LOG(B)
```

En la variable B debe encontrarse almacenado el valor de la base deseada. De esta forma, en LBX se obtiene el logaritmo en base B del número X.

ATN

Calcula el arco cuya tangente es el valor que aparece en su argumento.

Formato: ATN(<dato>)

Ejemplos:
```
20 LET ARCO=ATN(X)
70 PRINT ATN(A+B/2)
```

Matemáticas divertidas

Las funciones matemáticas expuestas serán de gran utilidad para la resolución de problemas sofisticados. Cualquier estudio avanzado, dentro de cualquier disciplina, desde la economía hasta la física, hace uso de los logaritmos y de los funciones trigonométricas. Sin embargo, no es necesario apelar a la alta

matemática para dar un ejemplo del uso de estas funciones. Incluso es posible conseguir efectos divertidos con las funciones estudiadas.

Por ejemplo, se puede simular la trayectoria de una pelota botando por medio de las funciones trigonométricas. Para ello se utilizará el coseno, aunque bien es cierto que se podría utilizar el seno con parecido resultado. Más concretamente, y a la vista de las respectivas representaciones gráficas, se hará uso del valor absoluto del coseno.

Al variar el valor de X en la expresión ABS (COS(X)) se van obteniendo diferentes datos. Estos datos pueden asimilarse a las distintas alturas que toma la pelota. Por medio del comando PRINT se puede plasmar esa trayectoria en la pantalla. Las coordenadas X e Y del movimiento se corresponderán con los dos parámetros del argumento del comando PRINT AT. Estas coordenadas coincidirán con X y ABS (COS(X)) respectivamente. El valor de X se modificará, obteniéndose así las diferentes posiciones. Como carácter a imprimir en las respectivas posiciones se utilizará la letra O por su similitud con una pelota. El cuerpo principal del programa lo constituirá, pues, la siguiente instrucción:

```
30 PRINT AT(X,ABS(COS(X))) "O"
```

Esta línea debe arroparse con un bucle FOR, que permita introducir la variación de X. Puesto que X corresponderá a la coordenada horizontal de AT (columna de texto en pantalla) sus valores límite no deben sobrepasar el número de columnas permitido por el equipo. En el ejemplo utilizaremos el número 30 como valor máximo de X, aunque el usuario puede cambiarlo por aquel que dé una mejor representación en su propia pantalla.

Por otra parte, es sabido que el valor del coseno se mantiene entre 1 y −1. Con lo cual su valor absoluto variará entre 0 y 1. Para conseguir una mayor amplitud en el movimiento vertical será necesario aumentar este rango de variación. Ello se realiza multiplicando dicha expresión por el número de filas que deba abarcar. Veinte es un número que dará buen resultado en la mayoría de los ordenadores. Al igual que ocurría con el tope de amplitud horizontal, el usuario puede elegir el valor que considere más propiado para las características de su ordenador:

```
20 FOR X=1 TO 30
30 PRINT AT(X,INT(20*ABS(COS(X))) )"O"
40 NEXT X
```

Como puede verse se ha calculado la parte entera del resultado de la expresión en la zona "y" del argumento de AT. Ello evitará posibles errores derivados de la exigencia de utilizar un número entero de filas de texto.

DEG

Permite el cálculo de las funciones trigonométricas expresando los ángulos en grados.

Formato: (número de línea) DEG

Ejemplos: 30 DEG

RAD

Especifica, en el cálculo de funciones trigonométricas, que los datos se dan en radianes.

Formato: (número de línea) RAD

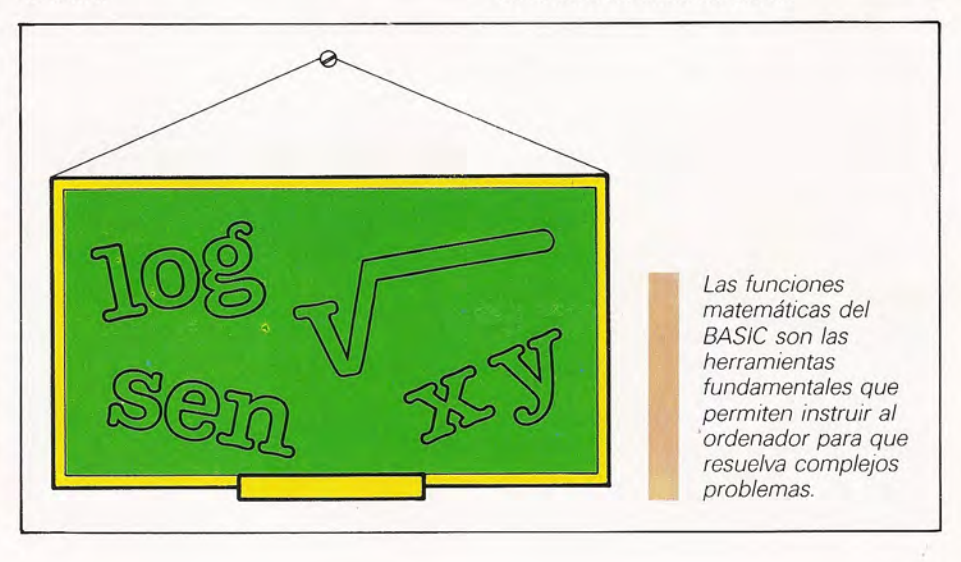
Ejemplos: 20 RAD

LOG

Calcula el valor del logaritmo neperiano del dato dado en su argumento.

Formato: LOG(<dato>)

Ejemplos:
```
10 LET LGX=LOG(X)
75 PRINT LOG(14*S+2)
```



Las funciones matemáticas del BASIC son las herramientas fundamentales que permiten instruir al ordenador para que resuelva complejos problemas.

TABLA DE CONVERSION					
Ordenador	**Conversión de ángulos**		**Logaritmos**		
	DEG	**RAD**	**LOG (<X>)**	**LOG10 (<X>)**	**EXP (<X>)**
AMSTRAD	DEG	RAD	LOG (<X>)	LOG10 (<X>)	EXP (<X>)
APPLE II (APPLESOFT)	—	—	LOG (<X>)	—	EXP (<X>)
APRICOT (M-BASIC)	—	—	LOG (<X>)	—	EXP (<X>)
ATARI	DEG	RAD	LOG (<X>)	CLOG (<X>)	EXP (<X>)
CBM 64	—	—	LOG (<X>)	—	EXP (<X>)
DRAGON	—	—	LOG (<X>)	—	EXP (<X>)
EQUIPOS MSX	—	—	LOG (<X>)	—	EXP (<X>)
HP-150	—	—	LOG (<X>)	—	EXP (<X>)
IBM PC	—	—	LOG (<X>)	—	EXP (<X>)
MPF	—	—	LOG (<X>)	—	EXP (<X>)
NCR DM-V (MS-BASIC)	—	—	LOG (<X>)	—	EXP (<X>)
NEW BRAIN	DEG (1)	RAD (1)	LOG (<X>)	—	EXP (<X>)
ORIC	—	—	LN (<X>)	LOG (<X>)	EXP (<X>)
SHARP MZ-700 (MZ-BASIC)	—	RAD (<X>) (2)	LN (<X>)	LOG (<X>)	EXP (<X>)
SINCLAIR QL	DEG (<X>) (2)	RAD (<X>) (2)	LN (<X>)	LOG10 (<X>)	—
SPECTRAVIDEO	—	—	LOG (<X>)	—	EXP (<X>)
ZX-SPECTRUM	—	—	LN (<X>)	—	EXP (<X>)

(1) Sólo aplicables en comandos gráficos. (2) Son funciones, devuelven el valor en radianes (RAD) o grados (DEG) del argumento (expresado en la unidad contraria).

Sólo falta poner una línea adicional con una instrucción de borrado de pantalla para que la trayectoria no quede enmascarada por el texto que pudiera existir en la pantalla. El programa completo adoptará el siguiente aspecto.

EXP

Calcula el valor del antilogaritmo del número incluido en su argumento. (e↑X).

Formato: EXP(<dato>)

Ejemplos: 30 LET ANTI=EXP(K)
120 PRINT EXP(—X/K)

```
10 REM BOLA 1
15 CLS
20 FOR X=1 TO 30
30 PRINT AT(X,INT(20*ABS(COS(X))) )"O"
40 NEXT X
50 END
```

AL ejecutar este primer programa, el resultado no es exactamente el deseado. El problema radica en que el valor del coseno se repite cada 6,28 (2x radianes) unidades de X). En consecuencia, su valor absoluto se repite con mayor frecuencia, exactamente cada unidades de X. De esta forma, al variar X

TABLA DE CONVERSION

Ordenador	Raíz cuadrada	Funciones trigonométricas					
	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	ASN (<X>)	ACS (<X>)	ATN (<X>)
AMSTRAD	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
APPLE II (APPLESOFT)	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
APRICOT (M-BASIC)	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
ATARI	SQR (<X>)	SIN (<X>)	COS (<X>)	—	—	—	ATN (<X>)
CBM 64	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
DRAGON	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
EQUIPOS MSX	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
HP-150	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
IBM PC	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
MPF	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
NCR DM-V (MS-BASIC)	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
NEW BRAIN	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	ASN (<X>)	ACS (<X>)	ATN (<X>)
ORIC	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
SHARP MZ-700 (MZ-BASIC)	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
SINCLAIR QL	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	ASIN (<X>)	ACOS (<X>)	ATAN (<X>)
SPECTRAVIDEO	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	—	—	ATN (<X>)
ZX-SPECTRUM	SQR (<X>)	SIN (<X>)	COS (<X>)	TAN (<X>)	ASN (<X>)	ACS (<X>)	ATN (<X>)

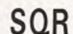
de 1 a 30, se efectúan casi 10 rebotes. Ello significa que un rebote se plasma en 3 columnas, con lo que el seguimiento de la trayectoria se hace muy difícil.

La solución consiste en hacer que cada rebote ocupe más espacio. Lo que equivale a ordenar que se produzca un menor número de ellos. La forma de conseguirlo se reduce a hacer que el argumento del coseno varíe dentro de unos límites inferiores. Para ello será suficiente con dividir el argumento del coseno por un número. Cuanto mayor sea este número, menos rebotes se producirán. Dividiendo por 3 la nueva versión del programa será la siguiente:

```
10 REM BOLA 2
15 CLS
20 FOR X=1 TO 30
30 PRINT AT(X,INT(20*ABS(COS(X/3))) )"O"
40 NEXT X
50 END
█
```

Esta vez sí que se aprecian los rebotes de la pelota. El único problema es que bota al revés. En efecto, se ve a la pelota caer hacia arriba y rebotar en el límite superior de la pantalla. Este efecto tiene su origen en el hecho de que la coordenada Y del comando PRIN AT aumenta hacia abajo. Es decir: valores mayores proporcionan alturas menores en la pantalla. En el caso que nos ocupa, la expresión situada como coordenada Y proporciona el valor de la altura «aumentando hacia arriba»... Es preciso darle la vuelta. Para ello, bastará con restar la citada expresión de su valor máximo:

```
20—INT(20*ABS(COS(X/3))).
```



SQR

Halla la raíz cuadrada del dato situado en su argumento.

Formato: SQR (<dato>)

Ejemplos:
```
10 LET A=SQR(17)
50 PRINT SQR(B+C)
```

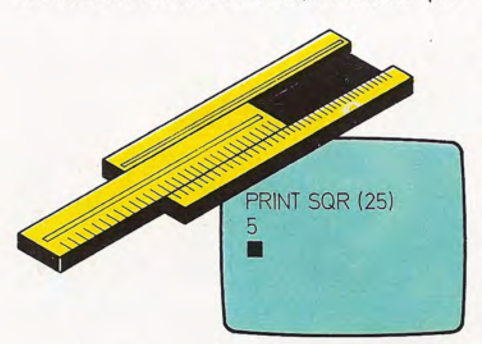
El listado correspondiente a la trayectoria puesta al derecho es el siguiente:

```
10 REM BOLA 3
15 CLS
20 FOR C=1 TO 30
30 PRINT AT(X,20—INT(20*ABS(COS(X/3)))  "O"
40 NEXT X
50 END
█
```

Aún puede realizarse una nueva mejora. Esta consistirá en introducir una amortiguación en los rebotes. Para ello se va a echar mano de otra de las funciones explicadas en este capítulo. La amortiguación a simular será de tipo exponencial, que responde a la siguiente fórmula:

<valor amortiguado>=<valor no amortiguado>*$e^{(-x/k)}$

En ella, X es la variable que irá tomando diferentes valores, mientras que



Con la ayuda de funciones de esta categoría, el ordenador será capaz de suplir a cualquier sofisticado utensilio manual de cálculo con eficacia y comodidad para el programador.

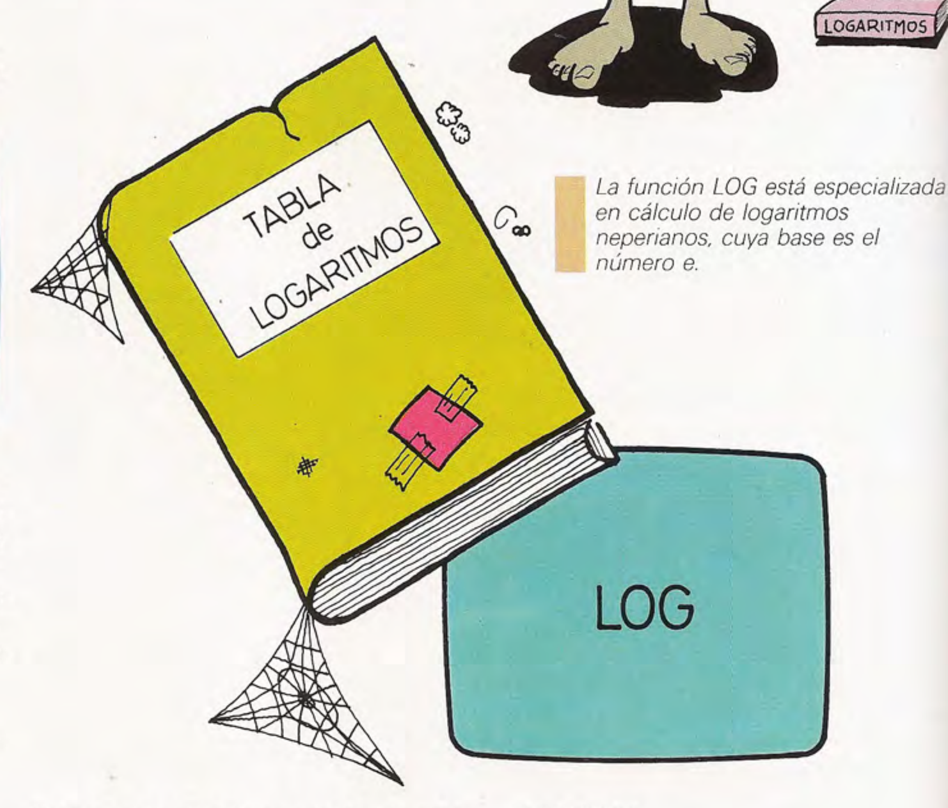
K es una constante que indica la velocidad de amortiguación. En el caso que nos ocupa, dicha fórmula adoptará el siguiente aspecto una vez codificada en BASIC

```
INT(20*ABS(COS(X/3))) *EXP(—X/K)
```

Aquí, el valor de K ha de ser especificado previamente. Para encontrar el valor ideal puede realizarse un estudio más profundo de la teoría matemática concerniente al hecho. Sin embargo, el camino más corto consiste en utilizar el ordenador para hacer pruebas. En primer lugar se dará el valor 1 a dicha constante para ver qué efecto produce. Así pues, el programa quedará definitivamente como sigue:

```
20 REM BOLA 4
15 CLS
20 FOR X=1 TO 30
30 PRINT  AT(X,20—INT(20*ABS(COS(X/3))) *EXP(—X/1)
   )"O"
40 NEXT X
50 END
```

El valor utilizado produce una amortiguación excesiva. La pelota, depués de golpear el suelo, no vuelve a ascender. Cambiando el valor de la constante K se conseguirán distintos grados de amortiguación. Algunos de estos valores pueden dar resultados plenamente satisfactorios; al respecto, las figuras muestran trayectorias correspondientes a los valores de K coincidentes con los números 15 y 30.



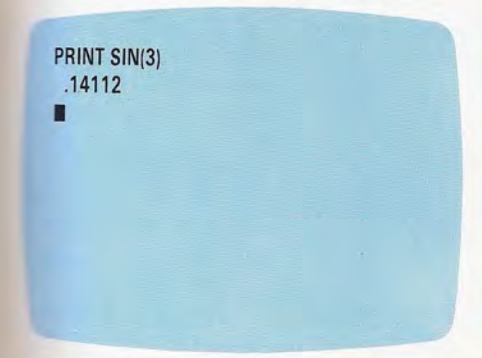
La función LOG está especializada en cálculo de logaritmos neperianos, cuya base es el número e.

Funciones a medida

Creación y uso de funciones definidas por el usuario



A la hora de construir un programa hay que echar mano de las palabras clave del BASIC. Estas, como se sabe, pueden ser comandos o funciones. Las funciones son aquellas palabras reservadas cuya ejecución proporciona un dato. Un ejemplo de función es SIN, cuyo cometido es calcular el valor del seno de un determinado dato numérico (valor del ángulo expresado en radianes). Un ejemplo de su modo de actuación es el siguiente:

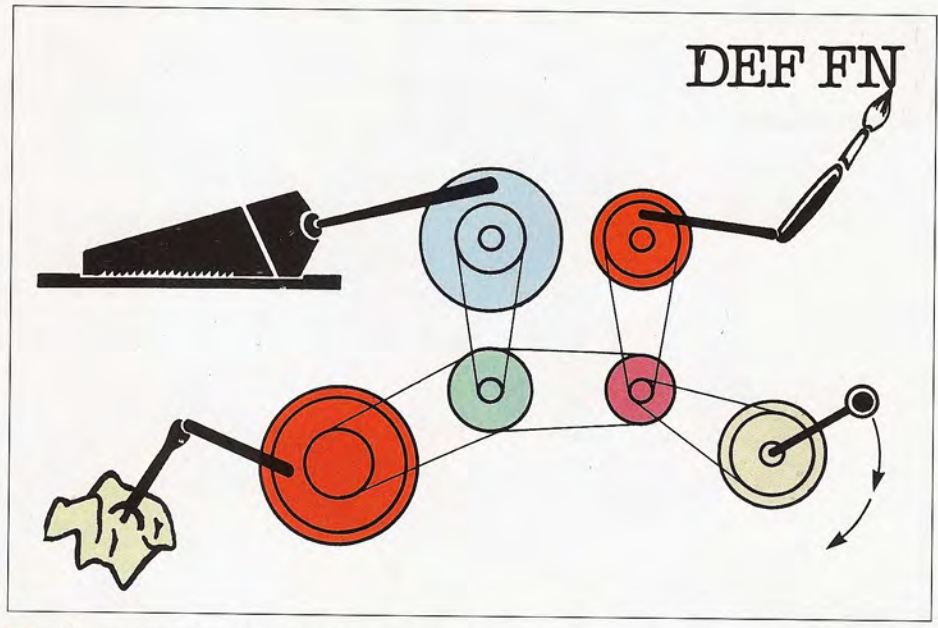


Normalmente, las funciones hacen uso de un dato, del que extraen la información para calcular el resultado pedido. En el ejemplo precedente, el dato es el número 3. Estos datos que son utilizados por las funciones reciben el apelativo de «datos de entrada». De la misma forma, los datos calculados por la función (resultados) reciben el nombre de «datos de salida».

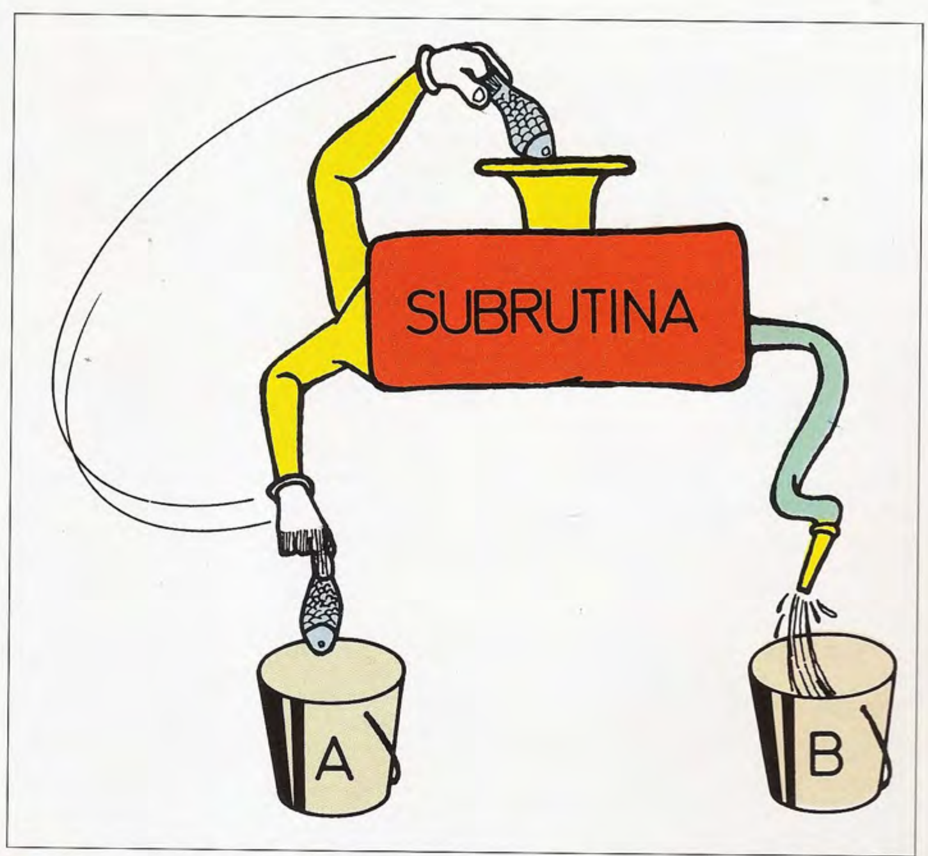
El uso de subrutinas

Los intérpretes del lenguaje BASIC suelen incluir un nutrido conjunto de funciones. Entre ellas se encuentran las ya estudiadas para el manejo de datos numéricos o alfanuméricos.

En ocasiones, puede resultar útil crear funciones específicas, no incluidas directamente en el traductor BASIC, para determinados cometidos. Por ejemplo, si se desea calcular reiteradamente el 10% de una serie de números, no es práctico repetir varias veces el mismo bloque de instrucciones. Para realizar dicho cálculo sobre un dato contenido

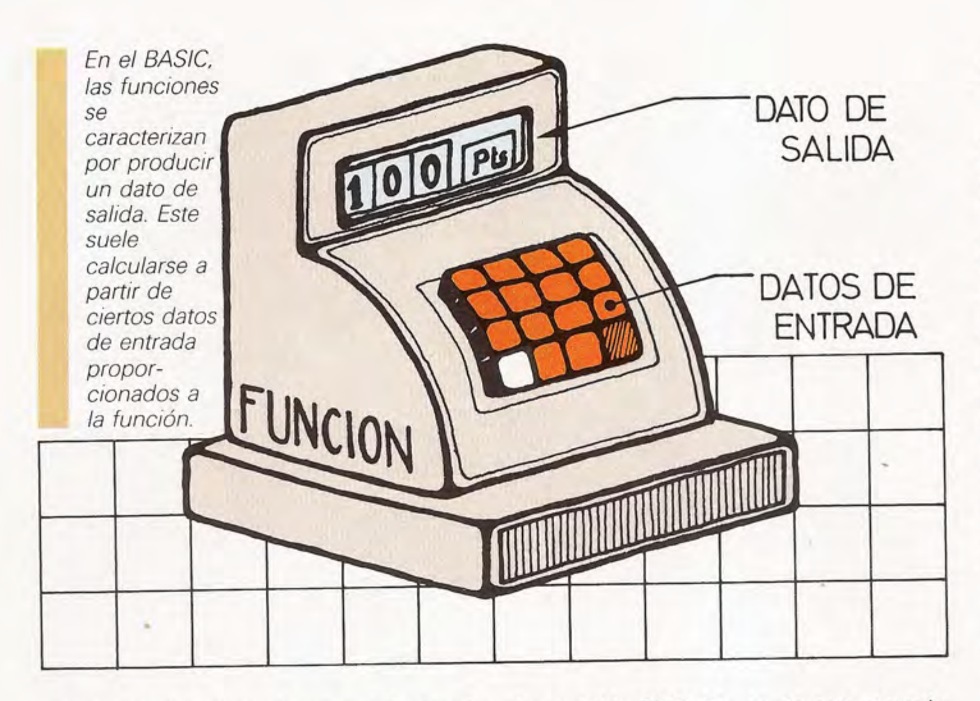


El comando DEF FN permite al usuario crear sus propias funciones.



Las funciones pueden simularse con el uso de subrutinas. Para ello, estas últimas han de recoger el dato de entrada de una variable asignada al efecto. El resultado también será depositado en una variable.

En el BASIC, las funciones se caracterizan por producir un dato de salida. Este suele calcularse a partir de ciertos datos de entrada proporcionados a la función.



en la variable X será necesaria la siguiente línea BASIC.

```
10 LET DPC=(X/100)*10
```

Al ejecutarla, el resultado se almacena en la variable DPC. Para evitar la continua repetición del contenido de esta línea se puede hacer uso de una subrutina. Una subrutina, como ya se comentó en un capítulo anterior, es un fragmento de programa que puede ser utilizado varias veces por el programa principal. La subrutina encargada de calcular el 10% tomaría este aspecto:

```
1000 LET DPC=(X/100)*10
1010 RETURN
```

En este caso, el dato de entrada de la «función 10%» se debe almacenar en la variable X. Una vez que se ha ejecutado la subrutina, el resultado aparecerá en la variable DPC. El uso de esta pseudofunción implicará tres pasos. El primero será situar el número base (del que se desea extraer el 10%) en la variable X. Después se ha de «llamar» a la subrutina; como quiera que ésta se encuentra en la línea 1000, se utilizará una instrucción GOSUB 1000. El último paso consiste en rescatar el resultado de la variable DPC. Estos tres pasos se plasmarían en BASIC de la siguiente forma:

```
10 LET X=15000
20 GOSUB 1000
30 PRINT DPC
```

Los referidos pasos habrán de ejecutarse cada vez que se quiera hacer uso de la subrutina.

Las funciones construidas con la ayuda de subrutinas son ligeramente engorrosas. Lo ideal sería disponer de funciones definibles que adoptaran el aspecto propio de las funciones que el BASIC ofrece dentro de su repertorio.

Veamos cuáles son las diferencias entre la función definida previamente y una función existente en el seno del intérprete; por ejemplo, la función seno.

Para hacer uso de la función seno basta con recurrir a la palabra clave SIN, mientras que la función definitiva necesita de un GOSUB 1000. El dato de entrada de SIN se adjunta como argumento de la función; es decir, acompaña a dicha palabra encerrado entre paréntesis. En el caso del tanto por ciento, es preciso realizar una asignación previa de la variable adecuada (X). Además, el resultado de la función SIN se obtiene automáticamente al utilizarla, mientras que el resultado del tanto por ciento se ha de extraer de una variable (DPC).

En el caso de la función implementada, basta con conocer el nombre de la misma y su utilización no implica la inclusión de nuevas líneas de programa; sencillamente, puede emplearse en una sentencia de asignación o como argumento de otro comando o función.

La función definida por medio de una subrutina necesita para su uso de más información. Es preciso conocer su localización: el número de línea donde se encuentra la subrutina. Además de ello, es imprescindible tener conocimiento de los nombres de las variables que utiliza (en el ejemplo X y DPC). Este método implica un cierto espacio en el listado; ni más ni menos, el dedicado a la consecución de los tres pasos arriba mencionados.

En BASIC existe otro método para la definición de funciones: se trata de una facilidad que incorpora el propio intérprete bajo los auspicios del comando DEF FN.

Definición de funciones

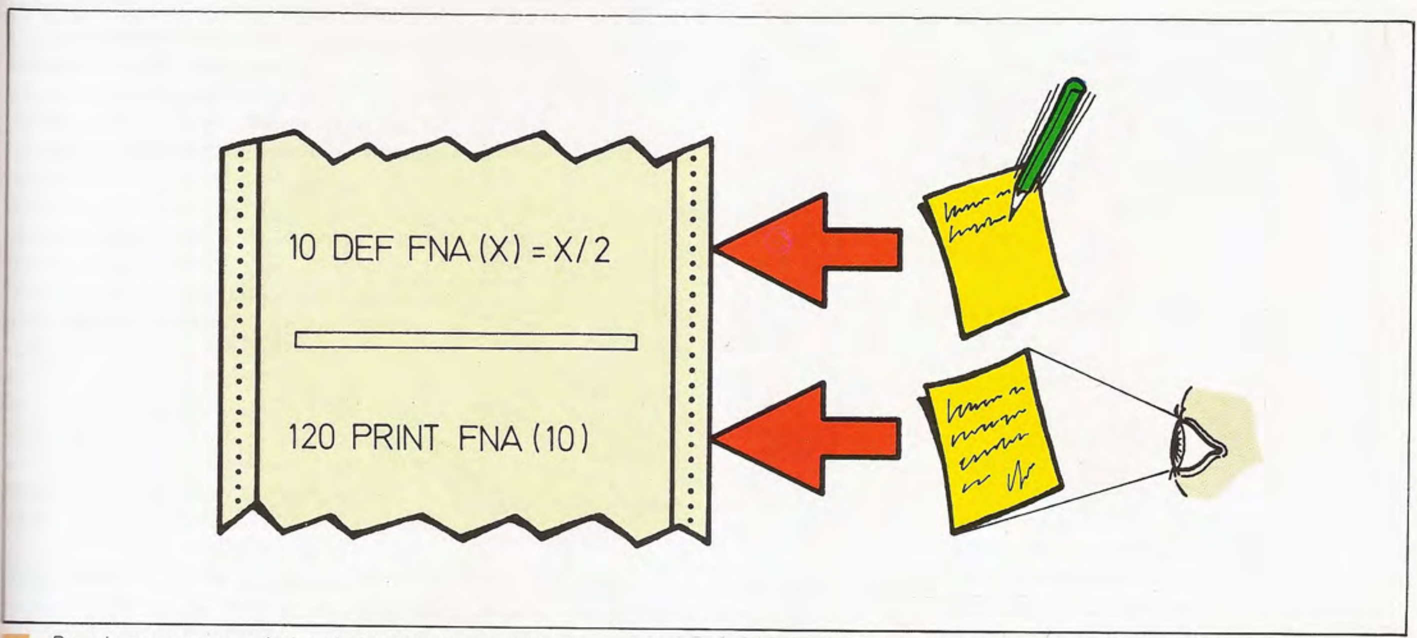
El comando DEF FN permite al usuario definir sus propias funciones: éstas servirán para aliviar la confección del resto del programa. Su ámbito de actuación se limita al programa en el que se han definido. Esto es, si se introduce un nuevo programa no se conservarán las antiguas funciones. La definición de una función de usuario se realiza en una sola línea de programa. En ella se de-

DEF FN

Define la función de usuario indicada por las instrucciones situadas en su argumento.

Formato: DEF FN<nombre>[(<parámetros>)]=<expresión>

Ejemplos:
```
10 DEF FNA(X)=SIN(X)+X^2
50 DEF FNPI=3.141592
```



Para la puesta en práctica de las funciones definidas por el usuario, son necesarios dos pasos. En el primero, el ordenador toma nota de la función definida por medio del comando DEF FN. Posteriormente, cuando ésta sea utilizada dentro del programa, la máquina revisará la lista de funciones definidas para aplicarla y calcular el correspondiente resultado.

ben incluir las palabras clave DEF y FN, contiguas y por ese orden; tras ellas, se escribe la definición de la función.

El formato de la definición de funciones muestra el siguiente aspecto:

(número de línea) DEF FN<nombre>
[(<variable>)]=<operaciones>

En dicha expresión cabe distinguir tres partes fundamentales, acompañando a las palabras DEF FN. Junto a la partícula FN se coloca un dato alfanumérico que servirá de nombre a la función. A continuación, y entre paréntesis, se sitúa un nombre de variable que se utilizará como dato de entrada de la función. Por último, y tras el signo «igual», se formulan las operaciones que calcularán el dato de salida.

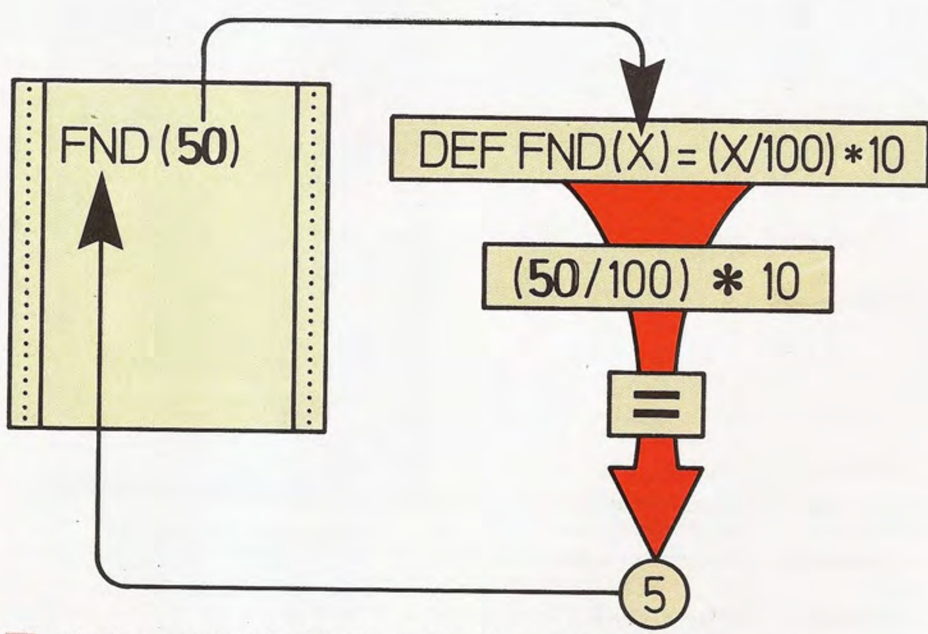
Un ejemplo de función definida es el siguiente:

```
10 DEF FND(X)=(X/100)*10
```

Se ha utilizado el mismo ejemplo que al principio del capítulo. En esta ocasión la función recibe el nombre «D». Ese es, precisamente, el carácter que acompaña a FN.

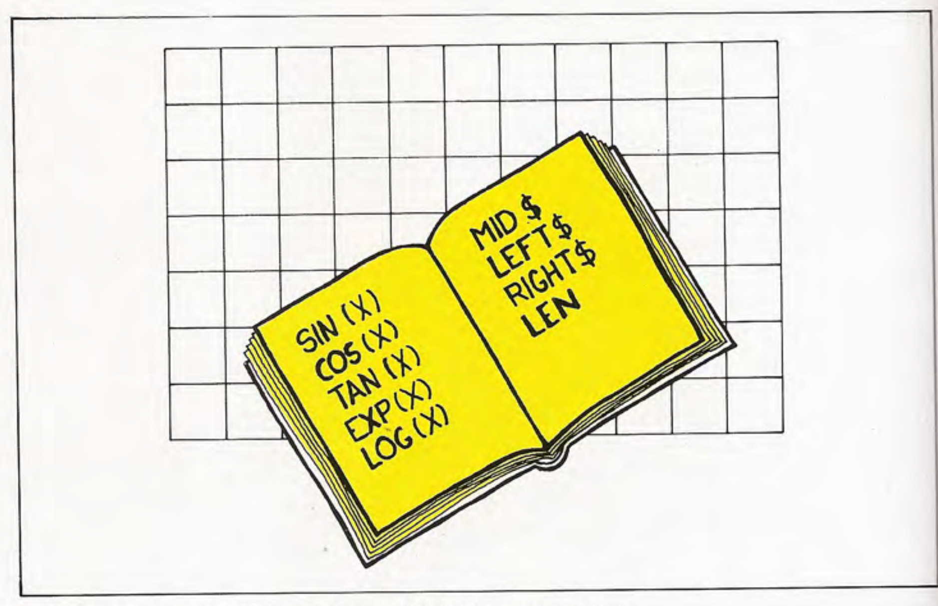
La variable de referencia, utilizada en el cuerpo de la definición es X. El nombre de variable, encerrado entre paréntesis, puede ser cualquiera. La única restricción está en que dicho nombre de variable debe coincidir con el que se emplee como dato de entrada en la definición. En el caso enunciado, X sirve para calcular el 10% de ese valor.

La variable referencial es una variable «ficticia». En efecto, no se trata es-



Al utilizar una función definida, el ordenador evalúa la expresión contenida en su definición. Para ello, sustituye la variable de entrada por el valor incluido como argumento de la función.

trictamente de una variable auténtica, sino que sólo sirve para indicar cuál de los datos de la definición es el de entrada. En el interior de la definición se pueden incluir más variables, cuyos valores serán aquellos que tengan asignados en el momento de hacer uso de la función.



La mayor parte de los dialectos BASIC incorporan una nutrida colección de funciones predefinidas.

Uso y disfrute de las funciones de usuario

Se ha estudiado ya el modo de definir una función de usuario. Este es el paso previo, pero sin duda la faceta más importante es la de su puesta en práctica.

Para acceder a una función definida hay que actuar de la misma forma que para acceder a una función propia del intérprete BASIC. En principio, se escribirá el nombre de la función (nombre otorgado en la definición de la misma); no obstante, para indicar que se trata de una función definida por el usuario, su nombre debe ir precedido de las letras FN. Esta sería la forma de utilizar la función definida en el apartado precedente:

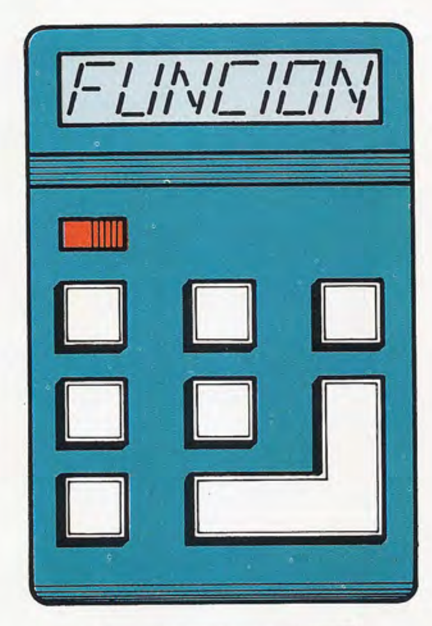


En el ejemplo se pone de manifiesto su semejanza respecto al uso de las funciones propias del intérprete BASIC. Al igual que con aquellas, el argumento indica el dato de entrada. Dicho argumento se sitúa entre paréntesis.

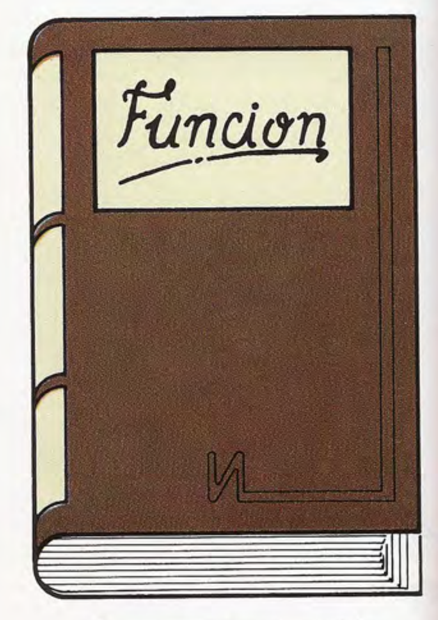
Para que sea posible utilizar una de estas funciones, es preciso definirla previamente. Ello implica dos cosas. Por una parte, ha de existir en el seno del programa una línea que defina la función por medio del comando DEF FN. Otro requisito es que el ordenador conozca dicha definición. Para ello la ejecución ha de pasar por la susodicha línea de definición, puesto que si no se ejecuta esa línea, la función no será utilizable, mostrando el aparato un mensaje de error.

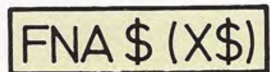
Biblioteca de funciones

Para redundar en el aspecto práctico, se analizan a continuación algunos









Habitualmente, los intérpretes BASIC permiten la definición de funciones adecuadas para trabajar tanto con datos numéricos como alfanuméricos. Estas últimas deben unir a su nombre el símbolo $

ejemplos de definición de funciones. En ellos se encuentran funciones no habituales en los intérpretes BASIC, así como otras de uso frecuente. Cada grupo de funciones puede resultar útil en distintos tipos de programa. Y todas ellas formarán un «cajón de sastre» o biblioteca de funciones diversas que convendrá tener a mano a la hora de ponerse a programar.

La mayor utilidad de las funciones reside en el cálculo de fórmulas matemáticas. Este es un hecho que vamos a considerar en nuestro primer grupo de funciones definidas, cuyo ámbito será la trigonometría. La mayor parte de los traductores BASIC disponen de cuatro funciones trigonométricas: seno (SIN), coseno (COS), tangente (TAN) y arco tangente (ATN). Remitiéndonos a un capítulo anterior, vamos a obtener ahora las restantes funciones trigonométricas; las primeras serán la secante, cosecante y cotangente.

```
10 DEF FNSEC(X)=1/COS(X)
20 DEF FNCSC(X)=1/SIN(X)
30 DEF FNCTG(X)=1/TAN(X)
```

La simplicidad de estas tres funciones obvia su explicación. Algo más complicadas resultan las funciones arcoseno y arcocoseno. Las fórmulas que las relacionan con ATN han sido ya comentadas en un capítulo anterior de la obra; éstas son:

```
ASN(X)=ATN(X/SQR(1—X^2))
ACS(X)=ATN(SQR(1—X^2)/X)
```

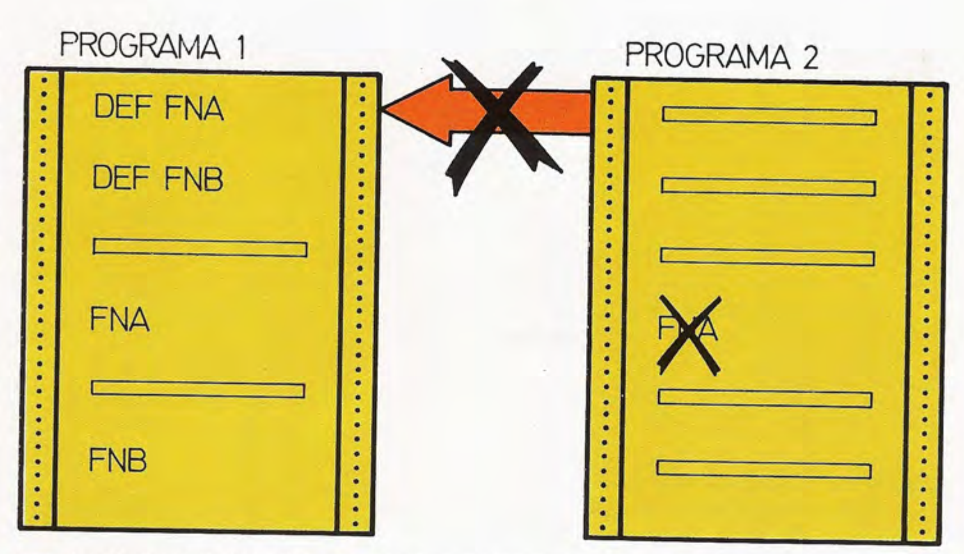
Su definición como funciones directas no tiene mayor complicación que la de copiar sus fórmulas en un comando DEF FN. Aquí está el resultado:

```
40 DEF FNAS(X)=ATN(X/SQR(1—X^2))
50 DEF FNAC(X)=ATN(SQR(1—X^2)/X)
```

Otras funciones útiles en numerosas ocasiones son las hiperbólicas. El seno y coseno hiperbólico responden a las siguientes fórmulas:

```
SH(X)=(EXP(X)—EXP(—X))/2
CH(X)=(EXP(X)+EXP(—X))/2
```

Según estas expresiones, se pueden crear muy fácilmente las funciones que las calculan:



Las funciones definidas en un programa pueden ser compartidas por otros programas. En consecuencia, para hacer uso de una función de esta índole, es obligado definirla dentro del mismo programa en el que vaya a ser utilizada.



En ocasiones puede ser necesario definir funciones con varios parámetros o datos de entrada. Este tipo de funciones permiten obtener un dato de salida o resultado dependiente de varios parámetros.

```
60 DEF FNSH(X)=(EXP(X)—EXP(—X))/2
70 DEF FNCH(X)=(EXP(X)+EXP(—X))/2
```

Para definir la tangente hiperbólica se hará uso del seno y coseno. La tangente es igual al cociente entre seno y coseno, con lo que su definición se puede reducir a una llamada a las funciones anteriores.

```
80 DEF FNTH(X)=FNSH(X)/FNCH(X)
```

Siguiendo con las funciones poco habituales vamos a entrar ahora en el

campo de los logaritmos. La primera función a definir es la que obtendrá directamente el logaritmo decimal. La fórmula a aplicar es la siguiente.

```
LOG 10(X)=LOG(X)/LOG(10)
```

En ella se hace uso de la función logaritmo neperiano (LOG) presente en la práctica totalidad de ordenadores. Su definición se reduce, pues, a la siguiente línea de programa:

```
90 DEF FNLOG10(X)—LOG(X)=LOG(10)
```

Una vez que se dispone de la función logaritmo decimal, nada resulta más fácil que sintetizar su función inversa: el antilogaritmo. El cálculo del antilogaritmo decimal se resume a elevar la base 10 al dato del que se desea conocer su antilogaritmo. Por lo tanto, la función que proporcione ese resultado será bien fácil de definir.

```
100 DEF FNAL10(X)=10^X
```

Como punto final a esta colección de funciones, se ofrece el listado correspondiente a la definición a las diez nuevas herramientas de cálculo:

```
 10 DEF FNSEC(X)=1/COS(X)
 20 DEF FNCSC(X)=1/SIN(X)
 30 DEF FNCTG(X)=1/TAN(X)
 40 DEF FNAS(X)=ATN(X/SQR(1—X^2))
 50 DEF FNAC(X)=ATN(SQR(1—X^2)/X)
 60 DEF FNSH(X)=(EXP(X)—EXP(—X))/2
 70 DEF FNCH(X)=(EXP(X)+EXP(—X))/2
 80 DEF FNTH(X)=FNSH(X)/FNCH(X)
 90 DEF FNLOG10(X)=LOG(X)/LOG(10)
100 DEF FNAL10(X)=10^X
```

Funciones no matemáticas

La definición de funciones no se limita a los cálculos matemáticos. También es posible la creación de funciones que traten con otros tipos de datos. Por ejemplo, se puede definir una función que extraiga la inicial de una cadena de caracteres. Este sería su aspecto:

```
10 DEF FNINI$(A$)=LEFT$(A$,1)
```

El uso de esta nueva función es similar al de cualquier otra función, sea o



Las funciones sin parámetros proporcionan un dato fijo. En consecuencia, su resultado será el mismo cada vez que sean utilizadas dentro del programa.

no de tipo matemático. La única salvedad aparece en que ahora se utilizan datos de cadena de caracteres. Esta circunstancia la revela la presencia del signo de «dólar» al final del nombre de la variable. Como el resultado ha de ser un nuevo dato alfanumérico, el nombre de la función ha de llevar también ese identificador de tipo.

En ciertos casos pueden incluso mezclarse datos de distinto tipo dentro de una misma función. Esto ha de hacerse observando la precaución de no aplicar funciones numéricas a datos no numéricos, y viceversa. Veamos un ejemplo.

La función que sigue sirve para codificar mensajes.

```
10 DEF FNCO$(A$)=CHR$(ASC(A$)+2)
```

Esta admite un dato alfanumérico consistente en una letra, de la que extrae su correspondiente código ASCII (por medio de ASC(A$)). A ese dato numérico le suma el valor 2, alterando su código en consecuencia, y entrega como dato de salida el carácter asociado al nuevo código. En definitiva, la función devuelve una letra relacionada con la propuesta como dato de entrada. Ello puede servir, por ejemplo, para codificar mensajes secretos.

El siguiente programa hace uso de la referida función para codificar una palabra:

```
10 DEF FNCO$(A$)=CHR$(ASC(A$)+2)
20 INPUT "MENSAJE";M$
30 LET C$=""
40 FOR I=1 TO LEN(M$)
50 C$=C$+FNCO$(MID$(M$,I,1))
60 NEXT I
70 PRINT C$
```

El mensaje se desglosa en los correspondientes caracteres; acción ésta que realiza la función MID$(M$,I,1) de la línea 50. A cada carácter se le aplica la función predefinida. Proceso que se repite hasta convertir todos los caracteres del mensaje inicial (FOR I=1 TO LEN(M$)). Los sucesivos caracteres codificados se van almacenando en la variable alfanumérica C$. Al final, se muestra el resultado obtenido en dicha variable.

La ejecución de este pequeño programa dará el siguiente resultado:

```
RUN<CR>
MENSAJE? PEPE
RGRG
█
```

En la ejecución se observa como el mensaje PEPE se transforma en RGRG. Esta clave (u otra parecida) puede ser útil para intercambiar mensajes secretos.

El programa adecuado para decodificar los mensajes en clave coincidirá con el anterior; la única diferencia aparece-

TABLA DE CONVERSION			
Ordenador	DEF FN		
	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=>exp.>
AMSTRAD	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
APPLE II (APPLESOFT)	—	—	—
APRICOT	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
ATARI	—	—	—
CBM64	DEF FN<n>(<p>)=<exp. num.>	—	—
DRAGON	DEF FN<n>(<p>)=<exp. num.>	—	—
EQUIPOS MSX	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
HP-150	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
IBM PC	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
MPF	DEF FN<n>(<p>)=<exp. num.>	—	—
NCR DM-V (MS)-BASIC	DEF FN<n>(<p>)=<exp. num.>	—	DEF FN<n>=<exp.>
NEW BRAIN	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
ORIC	DEF FN<n>(<p>)=<exp. num.>	—	—
SHARP MZ-700 (MZ-BASIC)	DEF FN<n>(<p>)=<exp. num.>	—	—
SINCLAIR QL	*	*	*
SPECTRAVIDEO	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>
ZX-SPECTRUM	DEF FN<n>(<p>)=<exp. num.>	DEF FN<n>(<p>)=<exp. cad.>	DEF FN<n>=<exp.>

<n>: nombre de la función. <p>: parámetro. <exp. num.>: expresión numérica. <exp. cad.>: expresión alfanumérica. DEF FN<n>=<exp.>: definición con número de parámetros variable. *: el QL permite definiciones en más de una línea.

rá en la función predefinida. La función de decodificación será la siguiente.

```
10 DEF FNDE$(A$)=CHR$(ASC(A$)—2)
```

En ella se realiza la acción contraria, restando dos unidades a cada código para restablecer el código real.



La función FNCO$ definida en el texto, sirve para codificar mensajes por medio de una clave secreta.

Funciones sin parámetros

En determinadas ocasiones se utilizan funciones en las que el dato de salida es fijo. Por ejemplo, en los equipos que no disponen de un comando para el borrado de pantalla, éste se puede generar por medio de una función:

```
10 DEF FNC$(X)=CHR$(26)
20 PRINT FNC$(0)
```

Esta sería la función adecuada para

aquellos casos en los que el borrado de pantalla está asociado al código ASCII 26. Si el código de borrado fuera otro, bastaría con cambiar dicho número.

Con esta función definida el borrado se logrará sin más que teclear PRINT FNC$(0). En nuestro ejemplo, el dato de entrada o parámetro de la función no interviene en absoluto; de ahí que no importe el dato que se incluya, puesto que éste no tiene repercusión alguna en la función. Realmente, cuando no se necesita parámetro de entrada, éste puede obviarse en la definición. Así pues, la función anterior se puede reducir en su expresión:

```
10 DEF FNC$=CHR$(26)
20 PRINT FNC$
```

Ahora, el borrado de pantalla resultará aún más cómodo: basta con teclear PRINT FNC$.

Las funciones sin parámetros pueden utilizarse, por ejemplo, para definir las constantes π (número PI) o *e* (número «e»).

Este es precisamente el cometido de las dos definiciones que siguen:

```
20 DEF FNPI=3.141592
30 DEF FNE=EXP(1)
```

Haciendo uso de la primera, el cálculo de la longitud de la circunfencia se resume a la siguiente función:

```
100 DEF FNL(R)=2*FNPI*R
```

¡Más parámetros!

Una función puede prescindir de parámetros, como ya se ha señalado. No obstante, también admite la situación opuesta: utilizar más de un parámetro. En tal caso, éstos se separan por medio de comas, dentro de la zona encerrada por paréntesis.

Las funciones con múltiples parámetros resultan útiles para programar fórmulas. Por ejemplo, para el cálculo de la velocidad media de un móvil partiendo del espacio recorrido y el tiempo invertido:

```
100 DEF FNV(E,T)=E/T
```

Desde luego, pueden crearse funciones bastante más complejas. La que sigue calcula las calorías producidas al pasar una corriente eléctrica por un conductor (Ley de Joule).

```
200 DEF FNCA(R,I,T)=.24*R*I^2*T
```

En los parámetros se especifican la resistencia del conductor (R), la intensidad de la corriente (I) y el tiempo transcurrido (T).

Ambito de utilización

Una vez definida una función, ésta será accesible durante el resto del programa. Por este motivo, es conveniente que la ejecución sea por las líneas de definición al principio del programa. Desde ese preciso instante ya serán utilizables. E incluso al detenerse la ejecución del programa puede hacerse uso de ellas en modo directo, fuera del programa. Ello supone que su definición sigue siendo válida. Sin embargo, las funciones definidas dentro de un programa no pueden ser utilizadas en otro distinto. La razón se encuentra en el hecho de que la ejecución de un comando RUN borra tanto las variables como las funciones definidas con anterioridad. En consecuencia, para que sea posible hacer uso de las funciones en un segundo programa será necesario volver a definirlas dentro del nuevo programa.

DEF FN en el Sinclair QL

Algunos dialectos BASIC se alejan de la norma en ciertos aspectos. Ello puede contribuir a una programación más potente, aunque también puede resultar más compleja. Este es el caso del BASIC del QL. El intérprete BASIC del QL (el denominado SUPERBASIC) permite una gran flexibilidad en la definición de funciones. El primer lugar, éstas no se ven limitadas a una línea de programa. Una definición puede ocupar varias líneas, siempre y cuando ello se indique. Las referidas líneas deben estar encabezadas por una en la que aparezcan las palabras reservadas DEF FN. Junto a estas dos palabras se sitúan el nombre y los habituales parámetros de la función. Las líneas siguientes son las que definen el cometido de la función. Una vez completada la definición, ésta se cierra con las palabras END DEF.

En el interior de la función es posible incluir todo tipo de comandos y funciones, incluso bucles de tipo FOR/NEXT; y también se pueden definir variables cuyo ámbito de utilización se limite a la propia función. Estas variables se denominan locales y se identifican con el prefijo LOC.

De los cálculos que se efectúen por medio de una función sólo se devolverá un dato como resultado. Para indicar el valor de salida que debe proporcionar la función se utiliza la partícula RET.

Las funciones definidas en este dialecto gozan de todas las propiedades generales: múltiples parámetros, manejo de cadenas, etc.

Como ejemplo de definición de funciones en este curioso dialecto, se incluye una función que calcula el factorial de un número (n!=1*2* ... *n).

```
10 DEF FN factorial (n)
20 LOC suma
30 LET suma=1
40 FOR I=1 TO n
50 suma=suma*I
60 END FOR n (equivalente a NEXT n.)
70 RET suma
80 END DEF
```



Al igual que en otros muchos aspectos, la definición de funciones con el super-BASIC del Sinclair QL muestra claras diferencias con respecto al procedimiento habitual.

El BASIC en acción

Un alto en el camino



Este punto es un buen momento para detenernos y proponer un pequeño ejercicio que ayude a fijar ideas. Se trata del diseño de un juego muy similar al tradicional juego de los barcos.

La superficie de juego es la misma: una retícula con cien casillas, con diez filas identificadas por las letras que van de la A a la J y de diez columnas numeradas del 1 al 10. Dentro de esta superficie se encuentra escondido el submarino al que se pretende hundir. Este submarino no tiene ninguna intención como es lógico, de hacernos la tarea fácil. Esa es la razón de que durante el juego se esté moviendo sin parar. Por nuestra parte disponemos de un sensor bastante perfeccionado. Este proporcionará una medida de la distancia a la que se encuentra el submarino tras efectuar el lanzamiento de una de las minas.

En primer lugar, resulta conveniente situar en pantalla una referencia para conocer los puntos del plano sobre los que está permitido lanzar las cargas de profundida. Para ello se utilizarán unas cabeceras de fila y columna. Las filas se distinguirán por las letras que van de la A a la J. Por su parte, las columnas se identificarán con un número del 1 al 10.

Para dibujar las cabeceras de las columnas es conveniente utilizar un bucle, lo que proporciona una solución bastante elegante. Como cabecera de la columna se escribirá el valor que toma en cada pasada la variable contadora del siguiente bucle:

```
20 FOR I=1 TO 10
30 PRINT AT (0,2*I+5);I;" ";
40 NEXT I
```

En el caso de que el ordenador con el que se esté trabajando no posea la función AT, el bucle construido puede ser sustituido por las siguientes líneas de programa:

```
20 PRINT SPC(5); FOR I=1 TO 10
30 PRINT I; " ";
40 NEXT I
```

En último extremo, si existiesen aún problemas de compatibilidad con el dialecto BASIC del ordenador que esté utilizando, la rutina podría reducirse a la siguiente línea de programa:



Se trata, sencillamente, de lanzar cargas hasta hundir el submarino... ¡Pero cuidado! Obtendrá mejor puntuación el jugador que haga blanco en menos intentos.

```
20 PRINT "1 2 3 4 5 6 7 8 9 10"
```

A continuación, hay que proceder a colocar los índices de las filas; tarea que puede realizarse de la siguiente forma:

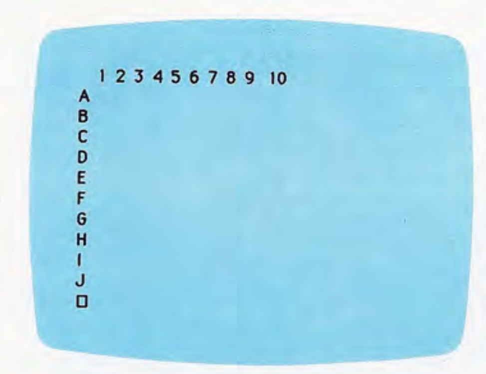
```
50 PRINT TAB(4); "A"
60 PRINT
70 PRINT TAB(4); "B"
80 PRINT
90 PRINT TAB(4); "C"
100 PRINT
110 PRINT TAB(4); "D"
120 PRINT
130 PRINT TAB(4); "E"
140 PRINT
150 PRINT TAB(4); "F"
160 PRINT
170 PRINT TAB(4); "G"
180 PRINT
190 PRINT TAB(4); "H"
200 PRINT
210 PRINT TAB(4); "I"
220 PRINT
230 PRINT TAB(4); "J"
```

Desde luego que la rutina propuesta es plenamente eficaz, aunque también es cierto que resulta muy pesada de escribir. Además, ocupa mucha memoria y es propensa a que se cometan errores a la hora de introducirla. En su lugar, cabe pensar en construirla en base a un bucle, de forma semejante al anterior. En este caso, no se pretende escribir un número sino una letra, por lo que la escritura directa del índice del contador no resulta una solución oportuna. Pero, si se piensa un poco, se llega a la conclusión de que estas letras consecutivas tienen además unos códigos ASCII también consecutivos. Dichos códigos ASCII van del 65 al 74 y, dado que es posible a partir del código ASCII generar el carácter correspondiente, lo que haremos será utilizar un bucle cuyo valor inicial sea 65 (es decir, el código ASCII de la primera de las letras a representar) y como valor final el 74 (código de la última de las letras a representar).



En pantalla se obtendría el resultado que puede observarse en la correspondiente figura.

Para medir los disparos o intentos de abatir el submarino, se utilizará una variable contadora; variable que es necesario inicializar a cero. La instrucción a ejecutar para poner el contador a cero



Pantalla de juego generada por la rutina que se incluye en el texto.

es la siguiente; por supuesto, en caso de que el contador coincida con la variable K:

```
100 LET K=0
```

A continuación, y dentro todavía del conjunto de instrucciones que sólo se ejecutarán una vez al principio del juego, queda por confeccionar una zona del programa que fije la posición de partida del submarino de forma aleatoria.

Aquí se hace necesario el empleo de la instrucción RND en orden a generar un número aleatorio. Desde luego, la función RND genera un número que en principio no servirá de mucho, puesto que se tratará de un número comprendido entre 0 y 1, pero sin alcanzar ninguno de estos límites. Por esta razón, se hace necesario efectuar un cambio de escala, convirtiendo el número generado en un valor comprendido entre uno y diez. La funcion capaz de obtener un número entre dos límites establecidos puede expresarse como sigue:

<número deseado>=INT (<número de posibilidades>*RND) + <valor mínimo del número>

En este punto hay que hacer la salvedad de que, probablemente, será necesario sustituir la instrucción RND en muchos ordenadores por la variante RND(0); ésta es la formulación correcta en muchos dialectos BASIC.

A continuación se entra ya en el juego propiamente dicho. De entrada hay que brindar al jugador la oportunidad de realizar su intento. Existen varias formas de instruir al ordenador para que pida al jugador su intento. La más sencilla consiste en ejecutar un INPUT en el que se pregunte por las coordenadas de lanzamiento. Estas se suministrarán de forma similar a como se hace en el tradicional juego de los barcos; esto es, mediante una letra y un número. La letra corresponderá a la fila y el número a la columna:

```
140 PRINT AT 23,5: INPUT "DONDE DISPARAS"; A$,M
```

Como quiera que la fila ha sido introducida mediante una letra, para saber a qué fila corresponde exactamente, el ordenador se verá obligado a convertir dicha letra en el número correspondiente. Ello supone obtener como resultado un número comprendido entre los valores 65 y 74. El rango del mismo debe estar situado de 1 a 10, luego basta tan sólo con restar 64 al código ASCII obtenido. De esta forma se tendra perfectamente definida la fila correspondiente al intento. Esta operación se puede realizar con la instrucción siguiente:

```
150 LET N=ASC (A$)—64
```

Una vez efectuado el lanzamiento, hay que determinar a qué distancia del blanco impactó la mina (evidentemente la distancia puede ser cero... ¡hundido!). La distancia no se medirá en diagonal, sino que, al menos en un principio, se evaluará de una forma un tanto curiosa aunque eficaz. Exactamente, se medirá la referida distancia en número de columnas desde el blanco al impacto y, análogamente, se medirá la separación en filas entre ambos puntos considerados (impacto y posición del submarino). Ambos valores se sumarán dando como resultado la información que se dará al jugador para que perfile su próximo lanzamiento.

Sobre el punto de la pantalla al que se dirigió el disparo aparecerá indicada la distancia a la que se encuentra el blanco. A su vez, tras cada andanada, se incrementará en una unidad el contador de lanzamientos efectuados. Todo ello es posible programarlo por medio de las siguientes instrucciones:

```
160 LET D=ABS(M—X)+ABS(N—Y)
170 PRINT AT N *2,M *2+5;D
180 LET K=K+1
```

El juego se desarrolla sobre una superficie de 100 posiciones o casillas; cada una de ellas se identifica por medio de su fila y columna.

La distancia al submarino viene dada como la suma de los valores de la diferencia entre la posición de éste y el punto de impacto del disparo, tanto en sentido horizontal (abcisas) como en el vertical (ordenadas).

$$D = X1 + Y1$$

En el caso de que el ordenador con el que se esté trabajando no posea la función AT, la línea 170 puede ser sustituida por la siguiente:

```
170 <HOME>:FOR I=1 TO N*2:PRINT:NEXT I:PRINT TAB(M
    *2+5); D
```

<HOME> adoptará en cada caso la formulación que corresponda a cada ordenador, y cuyo efecto sea llevar al cursor a la esquina superior izquierda sin borrar la pantalla.

A continuación se inspeccionará la distancia entre el blanco y el punto de destino del disparo, para determinar si ésta es nula (lo cual significará que se ha acertado en el blanco). Si el disparo ha dado en el blanco se producirá una bifurcación de la rutina final del programa. En ella se señala que el juego ha terminado y se muestra el número de andanadas que se han disparado. La instrucción necesaria para efectuar tal verificación puede ser la siguiente:

```
190 ID D=0 THEN GOTO 280
```

Si no se ha destruido al enemigo, el ordenador moverá el submarino y, de esta forma, dificultará la tarea del jugador. Para desplazar el blanco, la máquina debe generar un número aleatorio comprendido entre –1 y +1 para las columnas, y otro valor análogo para las filas; en función de dichos valores se actualizará la posición del submarino. Las siguientes instrucciones realizan esa acción:

```
200 LET X=Y+INT(3*RND)—1
210 LET Y=Y—INT(3*RND)—1
```

Puede darse el caso de que la trayectoria del submarino conduzca a éste fuera de los dominios del tablero de juego. En tal caso, es preciso corregirla simulando un rebote contra el límite del tablero. Las siguientes cuatro instrucciones se encargarán de corregir la trayectoria:

```
220 IF X=0 THEN X=2
230 IF X=11 THEN X=9
240 IF Y=0 THEN Y=2
250 IF Y=11 THEN Y=9
```

Una vez controlado el movimiento del submarino, hay que pasar a atender el próximo disparo que realice el jugador. Así pues, la secuencia del programa ha de regresar mediante una instrucción GOTO, a la línea 140 en donde se introducen las nuevas coordenadas de disparo:

```
260 GOTO 140
```

Por último tan sólo queda por construir la rutina final. En ella se indicará el fin de la partida y el número de intentos que han sido necesarios para hundir el submarino:

```
270 CLS
280 PRINT AT 8,8; "HUNDIDO EN ";K;" INTENTOS"
290 END
```

Y el programa está ya completo, ofreciendo el aspecto que revela el LISTADO 1.

El programa desde luego admite algunas mejoras. Una de ellas es la de incluir una rutina al final, que muestre el record obtenido hasta el momento por las personas que hayan jugado con el ordenador durante esa misma sesión. También puede incluirse una zona que pregunte al usuario si desea continuar jugando. Todo ello puede realizarse por medio de la rutina que pasamos a describir.

En primer lugar debe solicitarse el nombre de la persona que acaba de terminar el juego:

```
300 INPUT "INTRODUCE TU NOMBRE";B$
```

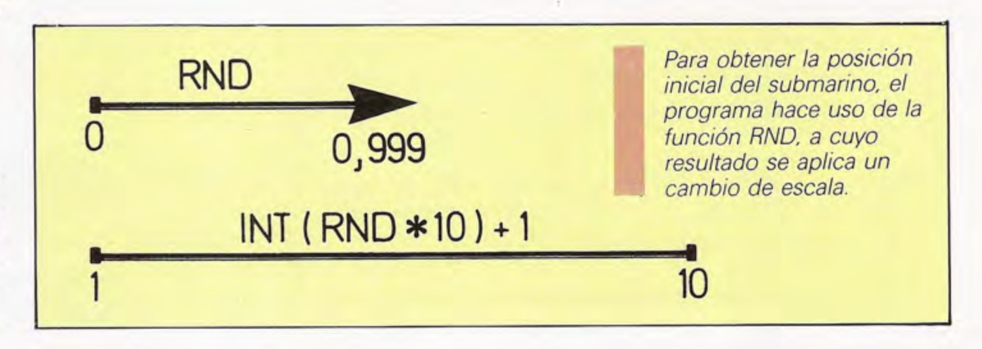
Dicho nombre queda pues almacenado en la variable de cadena B$. El nombre del jugador que hasta ese momento detente la máxima puntuación estará en R$. Las puntuaciones conseguidas se encuentran en las variables K (la del último jugador) y K1 (la del jugador que obtuvo la máxima puntuación). Para saber si el nuevo jugador ha obtenido una puntuación superior es necesario proceder a una comparación. En el caso de que el record se haya mejorado, se harán K1 y R$ iguales a K y B$, respectivamente.

```
310 IF K<K1 THEN LET K1=K:R$=B$
```

A continuación, se visualizará en la pantalla el nombre y la puntuación de la persona que ostente el record actual:

```
320 PRINT "EL RECORD ES";K1; " POR ",R$
```

Por último, nos encontramos con la zona encargada de preguntar al jugador



Para obtener la posición inicial del submarino, el programa hace uso de la función RND, a cuyo resultado se aplica un cambio de escala.

si desea realizar una nueva partida. Al efecto, debe proyectarse en la pantalla el correspondiente mensaje. De ello se ocupará la instrucción siguiente:

```
330 PRINT "QUIERES JUGAR DE NUEVO (S/N)?"
```

Para recoger la respuesta obtenida se utilizará la función INKEY$, asignando el valor de la misma a la variable C$. A continuación, se comprueba si se ha pulsado una tecla antes de continuar el proceso. En el caso de que no se haya pulsado tecla alguna, habrá que volver a examinar la entrada por el teclado hasta obtener una respuesta.

```
340 LET C$=INKEY$: IF C$=" " THEN GOTO 340
```

Para determinar si la respuesta es afirmativa o negativa entrarán en juego instrucciones condicionales. Estas detendrán el proceso en el caso de que la respuesta sea negativa, o bien lo reiniciarán si fuera afirmativa:

```
350 IF C$="S" THEN GOTO 10
360 IF C$="N" THEN END
```

Si la pulsación detectada no corresponde con las letras S o N, será preciso inspeccionar de nuevo el teclado hasta obtener una respuesta correcta:

```
370 GOTO 340
```

Por lo demás, es necesario añadir una línea de programa que inicialice las variables K1 y R$ con valores nulos para comenzar así el proceso. En el caso de la variable K1, es necesario inicializarla con un número relativamente alto, ya que la mejor puntuación es, en esta ocasión, la más baja. De inicializarla a 0 na-

VARIABLES UTILIZADAS EN EL PROGRAMA

A$	Dato introducido.
B$	Nombre del jugador en curso.
D	Distancia al blanco.
I	Indice de bucle.
K	Número de intentos.
K1	Puntuación record.
M,N	Posición del disparo.
R$	Nombre del jugador que obtuvo el récord.
X,Y	Posición del submarino.

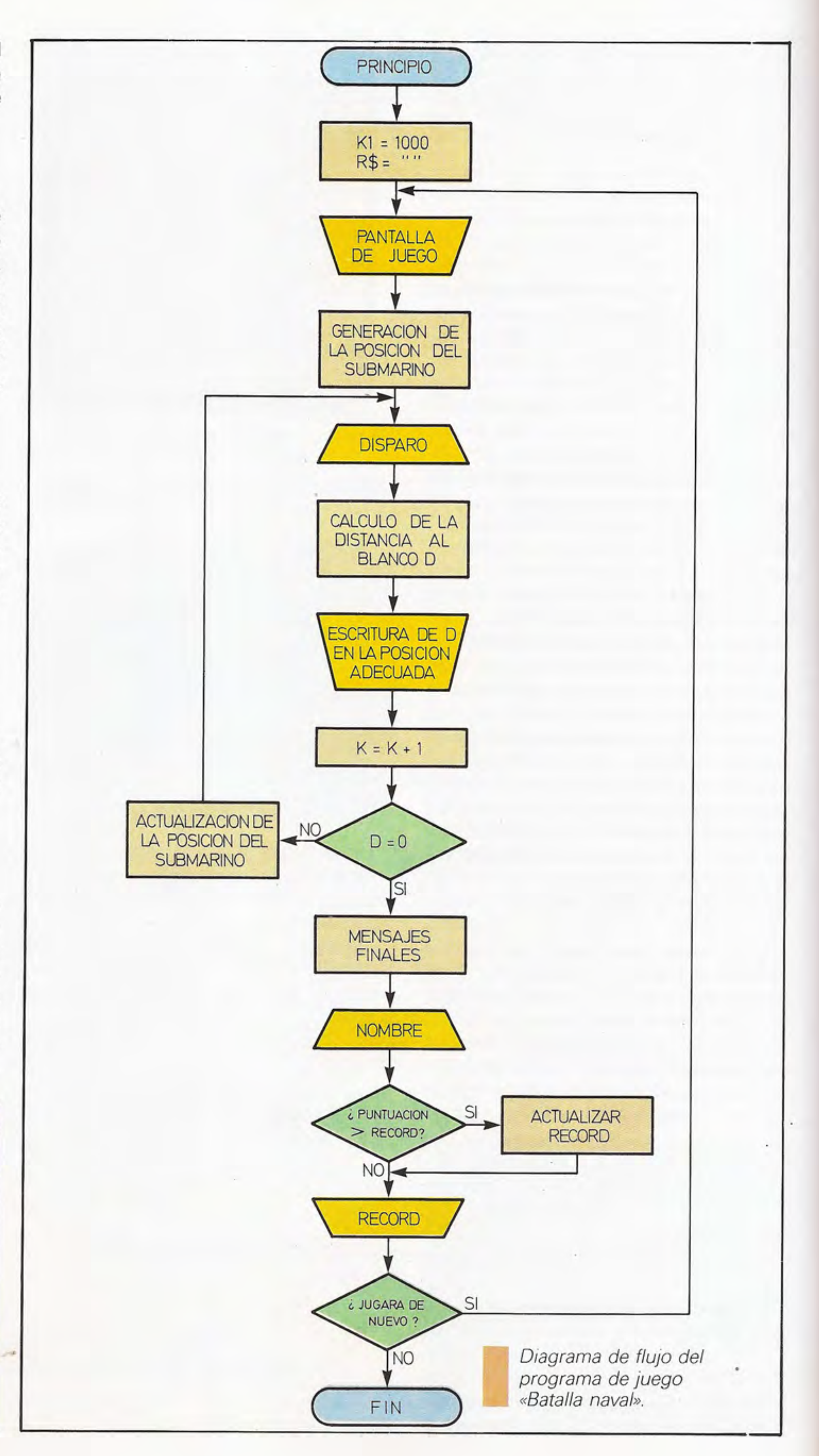


Diagrama de flujo del programa de juego «Batalla naval».

```
10 CLS
20 FOR I=1 TO 10
25 LOCATE 1,2*I+5
30 PRINT I;" "
40 NEXT I
50 REM CABECERAS
60 FOR I=65 TO 74
70 PRINT
80 PRINT TAB(4);CHR$(I)
90 NEXT I
100 K=0
105 REM POSICION DEL SUBMARINO
110 LET X=INT(10*RND(0)+1)
120 LET Y=INT(10*RND(0)+1)
130 REM INTRODUCCION DEL INTENTO
140 LOCATE 23,4:INPUT "DONDE DISPARAS";A$,M
150 LET N=ASC(A$)-64
160 LET D=ABS(M-X)+ABS(N-Y)
165 LOCATE N*2+1,M*2+5
170 PRINT D
180 LET K=K+1
190 IF D=0 THEN GOTO 280
200 LET X=X+INT(3*RND(0))-1
210 LET Y=Y+INT(3*RND(0))-1
220 IF X=0 THEN X=2
230 IF X=11 THEN X=9
240 IF Y=0 THEN Y=2
250 IF Y=11 THEN Y=9
260 GOTO 140
270 CLS
280 LOCATE 8,8:PRINT "HUNDIDO EN ";K;" INTENTOS"
290 END
```

LISTADO 1.
En el mismo se han sustituido los comandos PRINT AT por LOCATE/PRINT. Las líneas afectadas son 25/30, 140 y 165/170.

die sería capaz de obtener una mejor puntuación.

```
5 LET K1=1000:LET R$=" "
```

En ciertos ordenadores, su intérprete BASIC no incluye la función INKEY$, lo que obligará a recurrir a otro tipo de instrucciones. Por ejemplo, si estuviera disponible el comando GET, habría que sustituir la línea 340 por la siguiente:

```
340 GET C$:IF C$=" " THEN GOTO 340
```

Con ello el programa funcionará correctamente y sin ninguna diferencia respecto al anterior. No obstante, si tampoco existiera el referido comando en el dialecto BASIC utilizado, sería necesario recurrir a la instrucción INPUT.

```
5 K1=1000:R$=""
10 CLS
20 FOR I=1 TO 10
25 LOCATE 1,2*I+5
30 PRINT I;" "
40 NEXT I
50 REM CABECERAS
60 FOR I=65 TO 74
70 PRINT
80 PRINT TAB(4);CHR$(I)
90 NEXT I
100 K=0
105 REM POSICION DEL SUBMARINO
110 LET X=INT(10*RND(0)+1)
120 LET Y=INT(10*RND(0)+1)
130 REM INTRODUCCION DEL INTENTO
140 LOCATE 23,4:INPUT "DONDE DISPARAS";A$,M
150 LET N=ASC(A$)-64
160 LET D=ABS(M-X)+ABS(N-Y)
165 LOCATE N*2+1,M*2+5
170 PRINT D
180 LET K=K+1
190 IF D=0 THEN GOTO 280
200 LET X=X+INT(3*RND(0))-1
210 LET Y=Y+INT(3*RND(0))-1
220 IF X=0 THEN X=2
230 IF X=11 THEN X=9
240 IF Y=0 THEN Y=2
250 IF Y=11 THEN Y=9
260 GOTO 140
270 CLS
280 LOCATE 8,8:PRINT "HUNDIDO EN ";K;" INTENTOS"
290 REM RUTINA FINAL
300 INPUT "INTRODUCE TU NOMBRE ";B$
310 IF K<K1 THEN LET K1=K:R$=B$
320 PRINT "EL RECORD ES ";K1;" POR ";R$
330 PRINT "QUIERES JUGAR DE NUEVO (S/N)?"
340 LET C$=INKEY$:IF C$="" THEN GOTO 340
350 IF C$="S" THEN GOTO 10
360 IF C$="N" THEN END
370 GOTO 340
```

LISTADO 2.
El programa al completo. En el listado adjunto se utilizan de nuevo estructuras LOCATE/PRINT en sustitución de los comandos PRINT AT.

Con ella no basta con pulsar la tecla correspondiente a la respuesta, sino que además hay que accionar la tecla correspondiente al retorno de carro (RETURN o ENTER) tras introducir el valor oportuno.

```
340 INPUT C$
```

El listado completo, una vez efectuadas estas últimas correcciones, quedará tal como refleja el LISTADO 2 que acompaña al texto.

Para comprender más fácilmente la estructura y funcionamiento del programa confeccionado, es interesante examinar su correspondiente diagrama de flujo, el cual se reproduce en la figura adjunta.

Gráficos en BASIC

Introducción al dibujo en pantalla desde BASIC



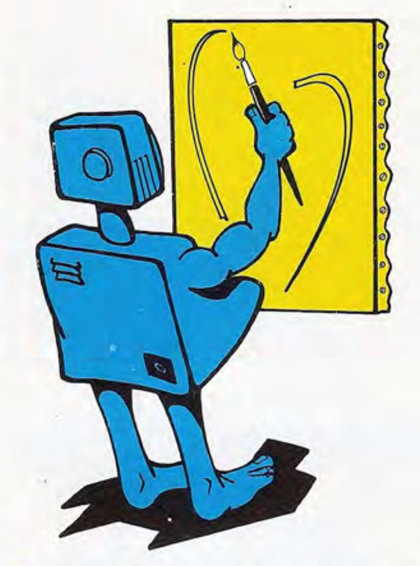
Una de las características más relevantes de los modernos ordenadores es su capacidad para crear gráficos. La representación gráfica de datos ayuda a su asimilación y a su mejor comprensión. Resulta más adecuado presentas las ventas de una empresa en un gráfico de barras que en una lista de números. De esta forma es mucho más sencillo observar la trayectoria de un simple vistazo.

Los gráficos por ordenador son imprescindibles en una larga serie de actividades: desde el diseño a los juegos, pasando por la educación. En general, una buena presentación hace más agradable el trabajo con el ordenador. Este es el motivo por el cual los fabricantes de ordenadores se preocupan cada vez maś de las posibilidades gráficas. Los más novedosos aparatos incorporan funciones impensables hace tan sólo unos años.

Desgraciadamente, la forma de tratar las posibilidades gráficas es muy diferente en cada ordenador. Cada fabricante ha tratado de mejorar las prestaciones que ofrecía la competencia, diversificándose así el modo de acometer esta materia. A lo largo del presente capítulo introductorio se seguirá la norma establecida por Microsoft; filosofía ésta extensiva a un amplio número de aparatos. En los restantes casos, los comandos BASIC se asemejan a los del BASIC-Microsoft, variando en algunas de sus características. Estas diferencias se indicarán a lo largo del texto y en la tabla de compatibilización que lo acompaña.

Resolución y modos gráficos

La característica que más influye en la diversidad de formas de crear gráficos es el «tamaño» de la pantalla. Las imágenes que aparecen en ésta, están formadas por un número de líneas horizontales. Cada una de estas líneas, a su vez, agrupa un número limitado de puntos. La imagen total se compone por la iluminación de los puntos adecuados.

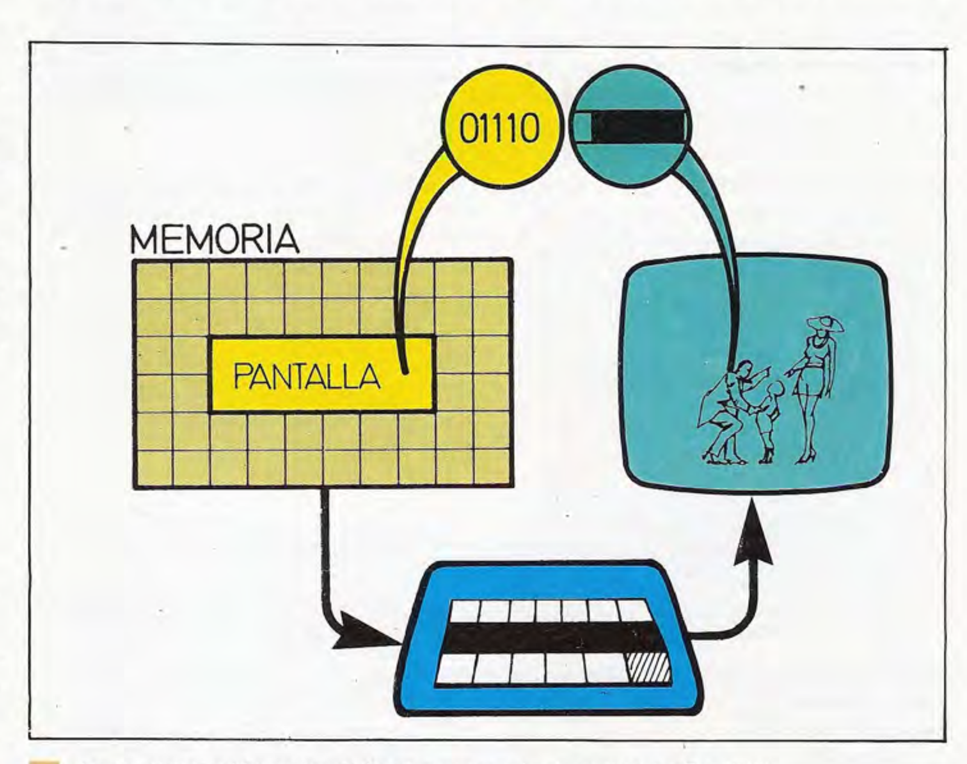


Los gráficos: una de las facultades más espectaculares del ordenador.

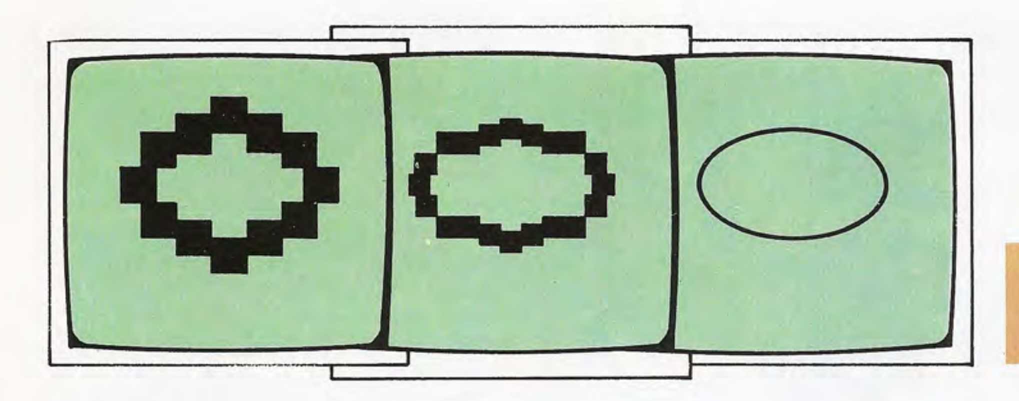
De ello se deduce que, a mayor cantidad de puntos, mejor definición de la imagen.

El número de puntos que es capaz de representar un ordenador se denomina «resolución» del mismo. La resolución máxima de un aparato se especifica mediante dos cantidades: una representa el número de líneas y la otra indica los puntos de cada línea. Así, una resolución de 192×256 significa que se tienen 192 líneas de 256 puntos. En ese caso se dispondría de un número total de puntos igual a 49152 (192*256=49152). Estos puntos suelen denominarse también «pixels» (contracción de PIcture ELementS).

Para la representación en pantalla, el ordenador hace uso de una parte de la memoria en la que se almacena el estado actual de cada punto. Si la imagen se va a representar en blanco y negro, se necesitará un bit por pixel. De esta forma, el bit a uno indica que el correspondiente punto está encendido. Un circuito específico del ordenador se encarga de leer la referida memoria de pantalla y trasportar su contenido a la pantalla. Conociendo la ubicación exacta de la zona de memoria de pantalla, ésta puede ser alterada con el uso de comandos POKE. Sin embargo, éste es un método definitivamente arduo y tedioso. En



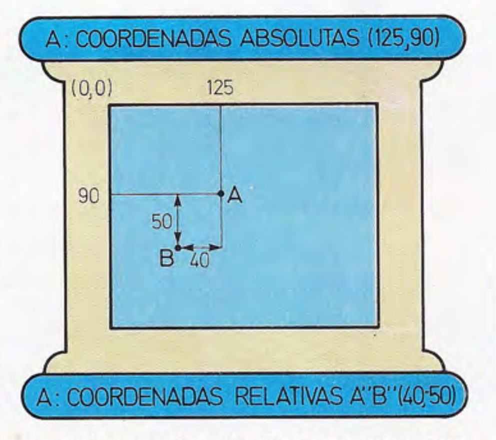
La representación en pantalla ocupa espacio de memoria. En dicho espacio se encuentra la información que utiliza el ordenador para iluminar los puntos que conforman la visualización.



Una característica a tener muy en cuenta a la hora de crear gráficos es el tamaño de la pantalla, medido en puntos luminosos, con el que trabaja el ordenador.

la mayoría de los casos, dicha alteración se ve facilitada por el empleo de los comandos gráficos del BASIC. Estos comandos son los que se describen en las páginas de este capítulo.

La necesidad de una zona de memoria dedicada a la representación en pantalla tiene otras implicaciones. Volvamos al ejemplo anterior. El tamaño de pantalla mencionado exige contar con 49152 bits. Un sencillo cálculo revela que ello representa un total de 6144 bytes (49152/8=6144). O lo que es lo mismo, 6144/1024=6 Kbytes. Esta es la ocupación de memoria en el caso de utilizar tan sólo dos colores (blanco=1, negro=0). Si se desean representar más colores será necesario contar con más memoria. Para cuatro colores no bastará con un bit por pixel; en tal caso serán necesarios al menos dos bits por cada punto a representar (00=negro, 01=rojo, 10=azul y 11=amarillo, por ejemplo). El resultado supone duplicar el espacio de memoria de pantalla para conservar la resolución. Las soluciones



La identificación de un punto puede realizarse especificando sus coordenadas absolutas (referidas al origen de coordenadas) o sus coordenadas relativas a otro punto de la pantalla.

tomadas por los diseñadores ante este problema son diversas.

En algunos aparatos la resolución y la memoria de pantalla son fijas. En otros se flexibiliza una de estas características para hacer un mejor uso de los recursos del aparato. En estos últimos casos se puede optar por dos métodos. Por una parte se puede aumentar la ocupación de memoria cuando se necesite más resolución o más colores. El otro método consiste en asignar un espacio fijo de memoria y jugar con la resolución y el número de colores. Este último implica una relación inversa entre resolución y colores: a mayor resolución menos colores disponibles, y viceversa.

El hecho de poder variar las características gráficas del aparato introduce un nuevo concepto: los modos gráficos. Los ordenadores que permiten al usuario elegir la resolución adecuada (dentro de unos límites) disponen de varios modos de representación o modos gráficos. El operador puede indicar el modo adecuado para la actividad en curso mediante un comando; comando que adopta formatos muy diferentes dependiendo del ordenador en cuestión. De ello se hablará más adelante.

CARACTERES ALFABETICOS
A B C D E F
CARACTERES NUMERICOS
1 2 3 4 5 6
CARACTERES GRAFICOS

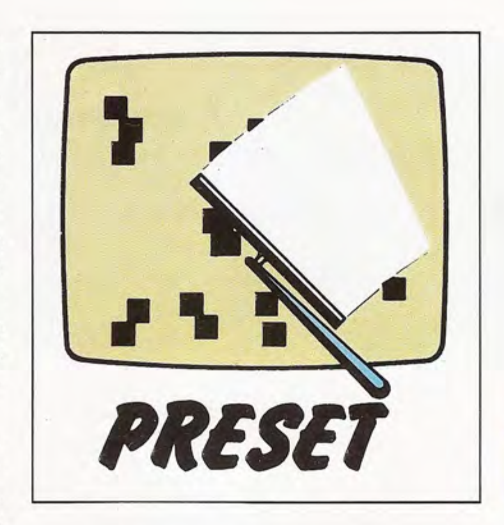
Algunos ordenadores poseen un repertorio de caracteres gráficos. Estos servirán para crear algunos dibujos elementales.

Baja resolución, caracteres gráficos

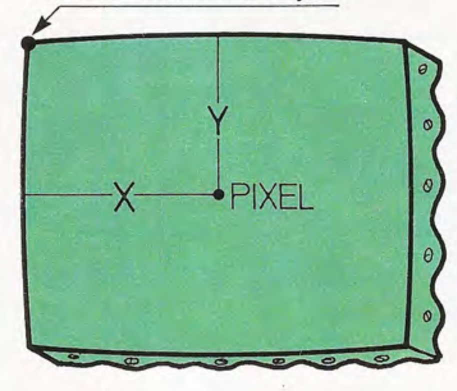
Se puede considerar como baja resolución la obtenida mediante el uso de caracteres. Al hablar del comando

PRINT y sus variantes, se subdividió a la pantalla en un determinado número de líneas y columnas. Estas indican la cantidad de texto representable en pantalla a un mismo tiempo. Ambos datos (número de líneas y de columnas) indican la «resolución en modo texto». Utilizando algunos caracteres especiales (como pueden ser el asterisco o los signos + y –) es posible llegar a construir gráficos elementales. Para ello se hará uso del comando PRINT AT (LOCATE, u otro similar). De esta forma se pueden obtener gráficos sin mucho detalle.

Algunos aparatos brindan un repertorio de caracteres denominados «caracteres gráficos». Estos son bloques especiales como cuadrados, líneas, arcos de circunferencia... con los que se pueden realizar algunos gráficos utilizando el mismo método que con los caracteres normales. La creación de dibujos con estos caracteres gráficos es similar a la construcción de una maqueta con piezas o bloques elementales. Con un poco de práctica se pueden realizar algunos dibujos, aunque con limitaciones.



El cometido del comando PRESET es el de borrar (apagar) puntos de la pantalla.



Cada punto de la pantalla o «pixel» queda plenamente identificado por medio de sus coordenadas.

Primeros pasos en alta resolución

La pantalla de alta resolución se puede concebir como un mosaico de puntos. Cada uno de esos diminutos puntos es un pixel. Y cada pixel puede ser identificado por sus dos coordenadas: vertical y horizontal.

El origen de estas coordenadas (el punto 0,0) variará de unos aparatos a otros. Por lo general, el origen se sitúa en una de las esquinas de la parte izquierda de la pantalla. El BASIC de Microsoft utiliza el ángulo superior izquierdo (éste será el que se emplee aquí); no obstante, será conveniente revisar en el manual de cada ordenador para la correcta situación de este punto.

Una vez identificado el origen, el primer paso consiste en iluminar uno de los pixels. A tal efecto se hace uso del comando PSET (en otros dialectos se emplea PLOT). Dicho comando necesitará las coordenadas para identificar el punto a iluminar. Estas se proporcionan en su argumento: primero la coordenada horizontal (X) y luego la vertical (Y). Este es el aspecto que muestra su formato:

(Número de línea) PSET (<X>, <Y>) [,<color>]

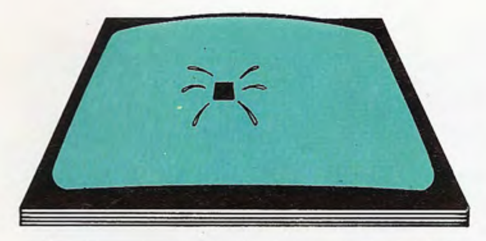
El tercer dato, suministrado opcionalmente, indica el color que ha de tomar el punto en cuestión. En aquellos ordenadores que disponen de varios modos de representación en pantalla, será necesario indicar el modo antes de ejecutar el comando PSET. Si no se hace así, se estará tratando de dibujar en la pantalla de texto, lo que puede producir un error de sintaxis. Si al ejecutar un comando PSET (o PLOT) aparece un mensaje de error, ello indicará que hay que ejecutar un comando de cambio de modo.

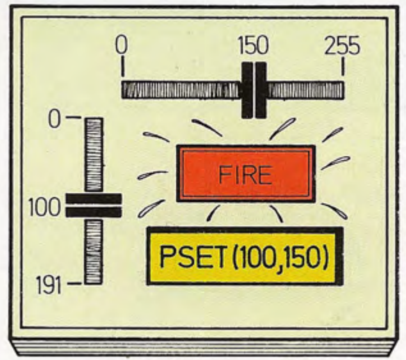
PSET

Ilumina el punto de la pantalla indicado por sus coordenadas.

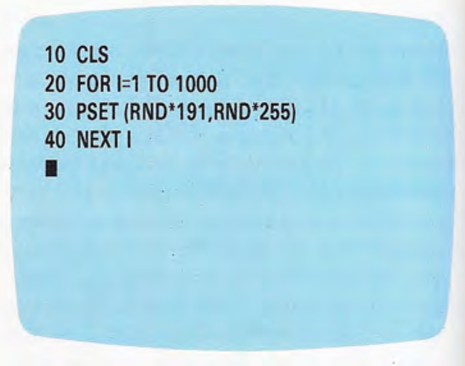
Formato: PSET [STEP] <X>, <Y>) [,<color]

Ejemplos:
```
10 PSET (100,150)
50 PSET STEP (10,—10)
```

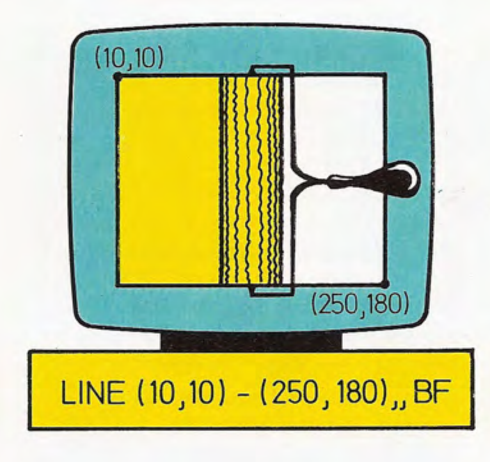




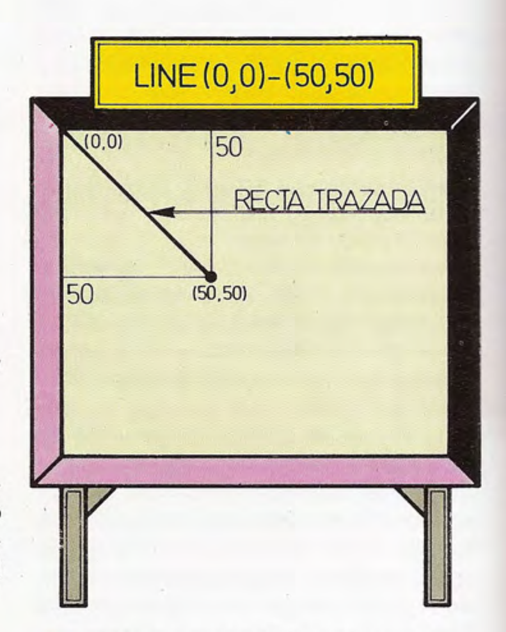
El comando PSET permite encender el punto de la pantalla que desee el usuario, sin más que especificar las coordenadas del mismo.



Este pequeño programa enciende puntos en lugares aleatorios de la pantalla. Al ejecutarlo se consigue un bonito cielo estrellado (si el fondo está en negro). Los ordenadores que utilizan varios modos de presentación gráfica necesitarán una línea previa en la que se especifique el modo gráfico adecuado. En algunos de estos aparatos se vuelve a la pantalla de texto al finalizar la ejecución del programa. Ello hará que desaparezcan los dibujos y se permita la visualización de caracteres. El programa anterior tomaría la siguiente forma en el BASIC de Microsoft.



El empleo conjunto de las opciones B y F permite crear rectángulos sólidos, esto es, rectángulos con toda la superficie interna coloreada.



Para el trazado rápido de rectas se hace uso del comando LINE. En su argumento se indican los puntos extremos del segmento a dibujar.

Variando el argumento de PSET se consiguen dibujar diferentes puntos. Siempre hay que prestar atención para no exceder los límites de la pantalla. En el caso de una pantalla de 192×256 pixels, la coordenada X puede variar entre 0 y 255, mientras que la vertical no debe exceder del valor 191. Con la resolución mencionada se puede crear el primer programa gráfico.

PRESET

Borra el punto de la pantalla que corresponde a las coordenadas indicadas en su argumento.

Formato: PRESET [STEP] (<X>,<Y>) [,<color>]

Ejemplos: 20 PRESET (125,125)
70 PRESET STEP (A,A/2)

```
5 SCREEN 2
10 CLS
20 FOR I=1 TO 1000
30 PSET (RND*191,RND*255)
40 NEXT I
50 GOTO 50
```

La línea 5 especifica el modo gráfico. En este caso se trata del modo de alta resolución. El comando empleado es SCREEN, el propio de Microsoft. Para mantener la imagen en pantalla se hace uso de la línea 50. Esta evita que se detenga la ejecución y se pierda el dibujo.

Con la ayuda de PSET (Point SET, poner punto) se pueden crear dibujos más complejos. Por ejemplo, se podría dibujar una línea; a fin de cuentas, una línea no es más que una sucesión de puntos.

El siguiente ejemplo muestra cómo trazar una línea horizontal.

```
10 INPUT "ALTURA";A
20 INPUT "LONGITUD";L
30 SCREEN 2
40 FOR I=1 TO L
50 PSET (I,A)
60 NEXT I
70 GOTO 70
█
```

Para dibujar la línea se ha tomado un dato fijo para la altura. Esta altura indica la coordenada Y de los puntos de la recta. Al tratarse de una línea horizontal, todos sus puntos tendrán la misma coordenada vertical; esta coordenada la introduce el operador en la línea 10. El otro dato que se le pide al usuario es la longitud del segmento a trazar. Este segundo dato se almacena en la variable L (línea 20). Ambos datos han de ser enteros, ya que el ordenador no puede trabajar con fracciones de pixel. Además, los datos proporcionados han de estar comprendidos dentro de los límites de la pantalla. De no ser así se produciría un error.

Las líneas de programa 40, 50 y 60 son las encargadas de representar la línea, cuyos extremos coincidirán con los puntos de coordenadas (1, A) y (L, A). El bucle FOR es el que se ocupa de incrementar la coordenada X desde el valor inicial al final.

Se ha visto la forma de iluminar puntos con PSET (Point SET). Pues bien, también existe la posibilidad de borrarlos. El comando encargado de esta función es PRESET (Point RESET, borrar punto). Su formato es muy parecido al de PSET, como puede apreciarse a continuación:

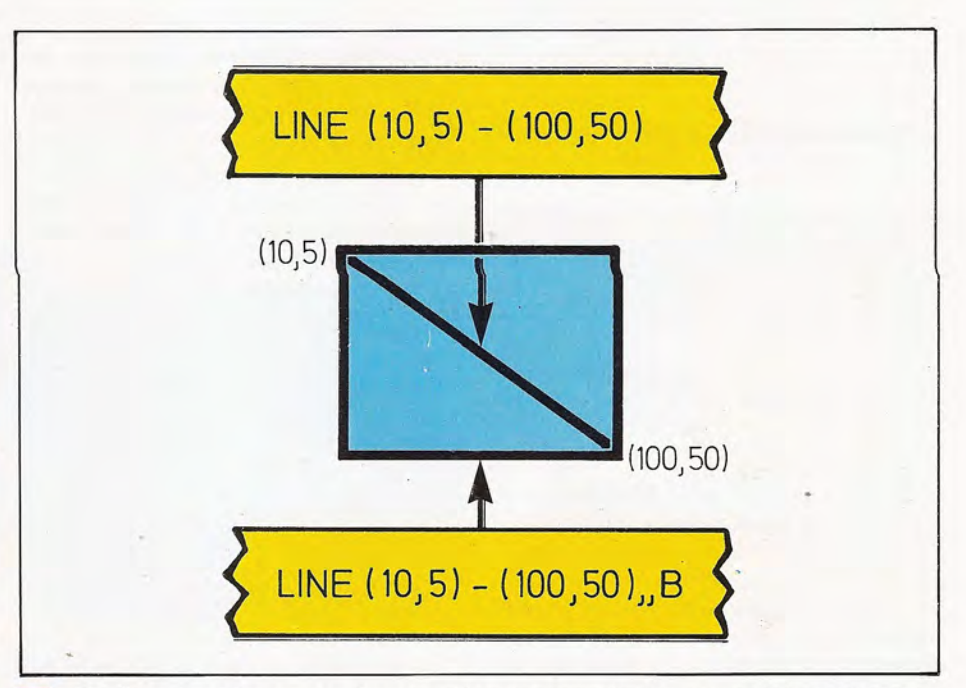
(Número de línea) PRESET (<X>, <Y>) [,<color>]

LINE

Traza una línea recta o un rectángulo entre los puntos especificados.

Formato: LINE [(<X1>,<Y1>)]–[STEP](<X2>,<Y2>)[,<color>[,B[F]]]

Ejemplos: 20 LINE (0,0)–(50,50),,BF



El comando LINE permite dibujar rectángulos con sólo hacer uso de la opción B. En la ilustración se observa un posible resultado.

Más líneas...

Más arriba se ha indicado un método para trazar líneas haciendo uso del comando PSET. Realmente, este método resulta superfluo en la mayoría de los ordenadores. Por regla general, se dispone de un comando BASIC que realiza ese mismo cometido. En el caso del BASIC de Microsoft, dicho comando es LINE (en otros intérpretes se identifica con DRAW). Por medio de LINE se pueden trazar líneas con suma facilidad:

(Número de línea) LINE (<punto inicial>)—(<punto final>) [,[<color>], [B][F]]

En las zonas de su formato identificadas como <punto inicial> y <punto final> se especificarán las coordenadas absolutas de ambos puntos. Dichas coordenadas se introducen separadas por una coma y en el orden habitual: primero X y luego Y.

Entre los restantes parámetros se encuentra el de color, el cual define el color que ha de tomar la línea a trazar. El color, como se verá más adelante, se especifica mediante un número o código.

Coordenadas absolutas y relativas

Se ha mencionado que en otros dialectos BASIC el trazado de rectas viene encomendado al comando DRAW. En la mayor parte de los casos éste utiliza un formato similar al siguiente:

(Número de línea) DRAW <incremento de X>, <incremento de Y>

En dicho formato llama la atención la presencia de tan sólo dos coordenadas. Con dos datos, tan sólo se puede especificar un punto. El otro punto que define la recta vendrá determinado por el comando que se haya ejecutado anteriormente. La filosofía es la siguiente: el último punto trazado servirá de punto de partida para el siguiente comando. En efecto, el punto siguiente se da especificando sus coordenadas relativas al punto actual. Ello significa que para determinar el punto final no es necesario conocer las coordenadas absolutas del mismo (las referidas al origen). Los datos que se han de suministrar dependen, pues, de la distancia al último punto. Como ejemplo, se muestra la realización de un cuadrado. Se supondrá que el último punto trazado corresponde al de coordenadas 0,0.

35

52

DRAW 52,35

Para dibujar una línea con el comando DRAW, hay que indicar las coordenadas relativas al último punto trazado.

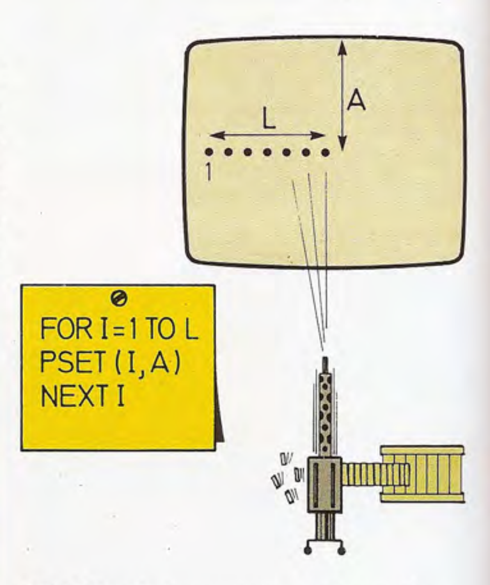
```
10 DRAW 50,0
20 DRAW 0,50
30 DRAW −50,0
40 DRAW 0, −50
█
```

Partiendo del punto 0,0 se traza una línea horizontal hasta el punto 50,0. Para ello se ha de incrementar la coordenada X en cincuenta unidades, dejando intacta la coordenada Y. Esta es, precisamente, la misión de la línea 10. Para trazar el lado que va hasta el punto 50,50 ha de incrementarse la coordenada Y (que estaba en cero) y mantener la X (que se ha actualizado a 50). Los restantes lados se trazan aplicando el mismo método. En ellos se aprecia la posibilidad de dar incrementos negativos, lo cual permite acceder a puntos cuya coordenadas sean inferiores a las del último punto referenciado.

A la vista del funcionamiento de las referencias relativas, cabe deducir que dicho método sólo permite trazar figuras conexas. Nada más lejos de la realidad. Se puede modificar la posición del último punto referenciado sin por ello trazar una recta; tal es el cometido del comando PSET (o PLOT en otros dialectos). Incluso, es posible indicar la situación del siguiente punto mediante sus coordenadas relativas a la posición actual.

Todo lo comentado es factible en el BASIC de Microsoft. La especificación de que el punto se expresa en sus coor-

Con el uso reiterado de PSET se pueden trazar líneas. Hay que recordar que, a fin de cuentas, una línea no es más que una sucesión de puntos.



DRAW

Traza una línea desde el anterior punto trazado al que se indica en su argumento.

Formato: DRAW <increm. X>,<increm. Y>

Ejemplos: 20 DRAW 50,50
70 DRAW LX,LY

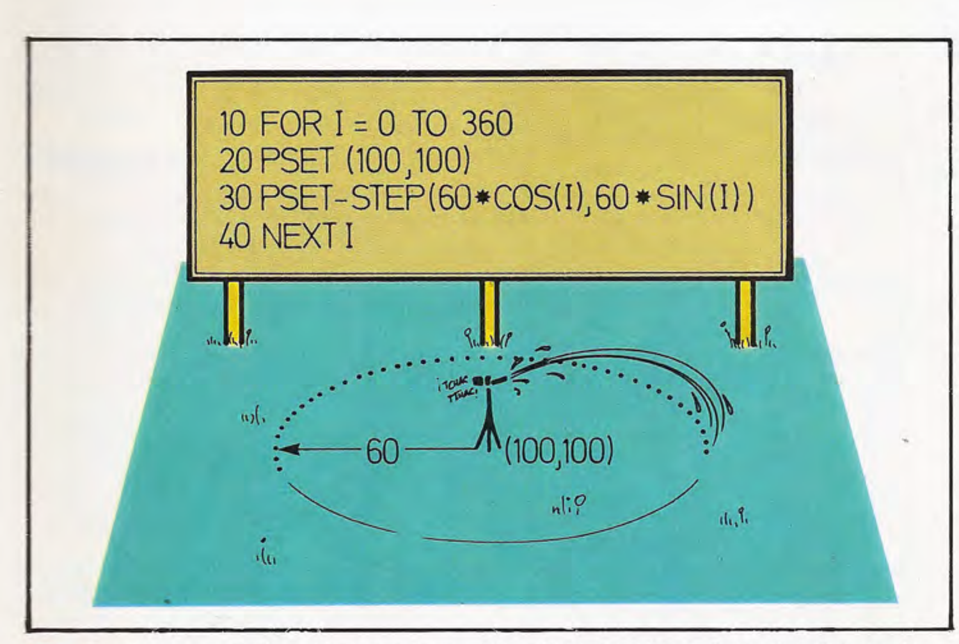
TABLA DE CONVERSION							
Ordenador	Puntos (Coordenadas absolutas)	Puntos (Coordenadas relativas)	Borrado de puntos	Rectas (Coordenadas absolutas)	Rectángulos (Contorno)	Rectángulos (Sólidos)	Rectas (Coordenadas relativas)
	PSET (<X>, <Y>)	**PSET STEP (<X>, <Y>)**	**PRESET (<X>,<Y>)**	**LINE (<P_1>)–(<P_2>)**	**LINE (<P_1>)–(<P_2>),B**	**LINE (<P_1>)–(<P_2>),BF**	**DRAW (<X>,<Y>)**
AMSTRAD[1]	PLOT <X>,<Y>	PLOTR <X>,<Y>	–	DRAW <X>,<Y>	–	–	DRAWR <X>,<Y>
APPLE II (APPLESOFT)	HPLOT <X>,<Y>	–	–	HPLOT <P_1> TO<P_2>	HPLOT <P_1> TO<P_2>TO...[2]	–	–
APRICOT (M-BASIC)	–	–	–	–	–	–	–
ATARI	PLOT <X>,<Y>	–	–	DRAWTO <P_2>[3]	–	–	–
CBM 64	–	–	–	–	–	–	–
DRAGON	PSET (<X>, <Y>,<C>	–	PRESET (<X>,<Y>)	LINE (<P_1>)–(<P_2>),<a>	LINE (<P_1>)–(<P_2>),<a>,BF	LINE (<P_1>)–(<P_2>),<a>,B	–
EQUIPOS MSX	PSET (<X>,<Y>)	PSET STEP (<X>,<Y>)	PRESET (<X>,<Y>)	LINE (<P_1>)–(<P_2>)	LINE (<P_2>)–(<P_2>),,B	LINE (<P_1>)–(<P_2>),,BF	–
HP-150	–	–	–	–	–	–	–
IBM PC	PSET (<X>,<Y>)	PSET STEP (<X>,<Y>)	PRESET (<X>,<Y>)	LINE (<P_1>)–(<P_2>)	LINE (<P_1>)–(<P_2>),,B	LINE (<P_1>)–(<P_2>),,BF	–
MPF	HPLOT <X>,<Y>	–	–	HPLOT <P_1> TO <P_2>	HPLOT <P_1> TO <P_2> TO...[2]	–	–
NCR DM-V (MS-BASIC)	–	–	–	–	–	–	–
NEW BRAIN	–	–	–	–	–	–	–
ORIC	–	–	–	–	–	–	DRAW <X>,<Y>,<C>
SHARP MZ-700 (MZ-BASIC)	SET <X>,<Y>	–	RESET <X>,<Y>	–	–	–	–
SINCLAIR QL	POINT <X>,<Y>	POINT_R <X>,<Y>	–	LINE <P_1> TO <P_2>	LINE <P_1> TO <P_2> TO...[2]	–	LINE_R TO <X>,<Y>
SPECTRAVIDEO	PSET (<X>,<Y>)	PSET STEP (<X>,<Y>)	PRESET (<X>,<Y>)	LINE (<P_1>)–(<P_2>)	LINE (<P_1>)–(<P_2>),,B	LINE (<P_1>)–(<P_1>),,BF	–
ZX-SPECTRUM	PLOT <X>,<Y>	–	–	–	–	–	DRAW <X>,<Y>

<X>: Coordenada horizontal del punto a trazar. <Y>: Coordenada vertical del punto a trazar. <C>: Código del color que ha de tomar el punto. <a>: Adoptará los valores PSET (para trazar) o PRESET (para borrar). <P_1> y <P_2>: Puntos inicial y final de la recta, dados en forma <X>, <Y>.

[1] Todos los comandos de Amstrad llevan otros dos parámetros relacionados con el color.

[2] El trazado de rectángulos se realiza por la concatenación de rectas en un mismo comando.

[3] La línea se traza desde el último punto referenciado hasta <P_2>. Este último se dan en coordenadas absolutas.



En el texto se explica cómo crear una circunferencia a base de puntos. Los puntos se definen por medio de sus coordenadas relativas al punto central.

denadas relativas se hace utilizando la cláusula STEP. Véase un ejemplo:

```
10 LINE—STEP (50,0)
20 LINE—STEP (0,50)
30 LINE—STEP (—50,0)
40 LINE—STEP (0,—50)
```

La presencia de la palabra STEP delante de los paréntesis indica que las coordenadas que se adjuntan están dadas en forma relativa. En el último ejemplo propuesto se vuelve a trazar el mismo cuadrado confeccionado con las rutinas anteriores.

La partícula STEP se puede emplear también con el comando PSET; en cuyo caso se tomará como origen el último punto trazado, siendo las coordenadas que se indiquen relativas a dicho punto.

Como ejemplo se trazará una circunferencia; para lo cual es preciso conocer la expresión de los puntos de la circunferencia en función de las coordenadas de su centro. Las fórmulas son las siguientes:

$X=X_0+R * \cos(a)$
$Y=Y_0+R * \text{sen}(a)$

En donde X_0 e Y_0 son las coordenadas del centro, R es la longitud del radio y «a» es el ángulo que forma el radio de cada punto con la horizontal. Como se trata de dibujar la circunferencia completa, se hará variar el ángulo de 0 a 360 grados (o de 0 a 2*PI radianes). He aquí el programa.

```
10 FOR I=0 TO 360
20 PSET (100,100)
30 PSET—STEP (60*COS(I),60*SIN(I))
40 NEXT I
```

Si el aparato procesa los ángulos medidos en radianes, la línea 10 habrá de sustituirse por la siguiente:

```
10 FOR I=0 TO 6.28 STEP 0.01
```

En el ejemplo se ha situado el centro de la circunferencia en el punto 100,100. El radio tomado en este caso es de 60 unidades (60 pixels). El comando de la línea 20 ilumina una y otra vez el centro de la circunferencia, lo cual permite al siguiente comando apoyarse en dicho punto. De esta forma, los puntos trazados con el comando de la línea 30 vienen referidos al punto central.

Cuadrados y rectángulos

Anteriormente se ha utilizado el comando LINE para dibujar un cuadrado. En realidad, LINE tiene más posibilidades: las relacionadas con los parámetros opcionales B y F. El primero de ellos permite dibujar rectángulos que tendrán siempre sus lados paralelos a los límites de la pantalla; esto es, verticales y horizontales, pero nunca oblicuos. Para trazar una de estas figuras basta con incluir una B en el argumento del comando LINE. El rectángulo vendrá definido por los dos puntos que se especifiquen; puntos que corresponderán a dos vértices opuestos de dicho rectángulo. Utilizando esta opción, el cuadrado del ejemplo anterior se obtiene con una sola línea de programa:

```
10 LINE (0,0)—(50,50),,B
```

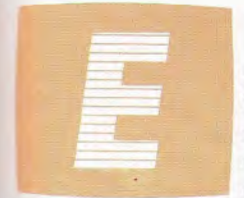
Las dos comas que preceden al parámetro de B revelan que el color no se ha especificado. Para emplear un color determinado habrá que especificar su correspondiente código entre ambas comas.

Existe una posibilidad adicional especificada con la opción F. La anterior hacía referencia a rectángulos (B de Box, caja). Esta otra permite rellenarlos (F de Fill, rellenar). Para utilizar la opción F es necesario trazar un rectángulo, no una recta. Con el empleo de BF aparecerá en pantalla un rectángulo sólido. Es decir, no sólo se trazará su contorno, sino que se rellenará toda su superficie con el color indicado. Así pues, el mismo cuadrado, aunque esta vez relleno, se puede crear con la rutina que sigue:

```
10 INPUT "COORDENADA X";X
20 INPUT "COORDENADA Y";Y
30 SCREEN 2
40 LINE (X,Y)—STEP (50,50),,BF
100 GOTO 100
```

La pantalla como lienzo

Gráficos en color



n este capítulo se introducen los comandos adecuados para el uso del color. Junto con ellos se presenta el encargado de trazar circunferencias. Todo ello permite la creación de los más complejos dibujos en la pantalla del ordenador.

Trazando líneas curvas

En el capítulo anterior se mostró la forma de trazar rectas "punto a punto"; método que permite también el trazado de circunferencias. Allí se mostraba un programa que permitía dibujar una circunferencia.

Sin embargo, en muchos aparatos existe un comando BASIC especializado en estas labores. Este comando responde a la palabra clave CIRCLE.

Para definir una circunferencia basta con dar su centro y su radio. De la misma forma, el comando CIRCLE admite estos dos datos en su argumento. El primero de ellos, el centro, se indica mediante sus dos coordenadas: horizontal y vertical (para localizarlas adecuadamente es preciso conocer la resolución que ofrece el aparato y el origen de coordenadas que toma). El dato correspondiente al radio se mide en "pixels" o puntos luminosos, siendo por lo tanto un número entero.

En consecuencia, la instrucción BASIC que produce el trazado de una circunferencia de radio 40 y centro en el punto 50,50 es la que sigue:

```
100 CIRCLE (50,50),40
```

La formulación elegida es la del BASIC de Microsoft. Para establecer su equivalencia en otros dialectos es conveniente remitirse a la tabla de conversión incluida en estas páginas.

En la ejecución del comando CIRCLE es preciso tener en cuenta las dimensiones de la pantalla. Si durante el trazado de la circunferencia se exceden los límites de ésta, se producirá un error y se detendrá la ejecución del programa.

Arcos de circunferencia

Las circunferencias no son lo único que se puede trazar directamente con el comando CIRCLE. Este comando permite también dibujar arcos, es decir, fragmentos de circunfernecia. Para ello se necesitan dos datos más: los que indican el principio y el final del arco. En este punto es necesario conocer cómo se recorre la circunferencia.

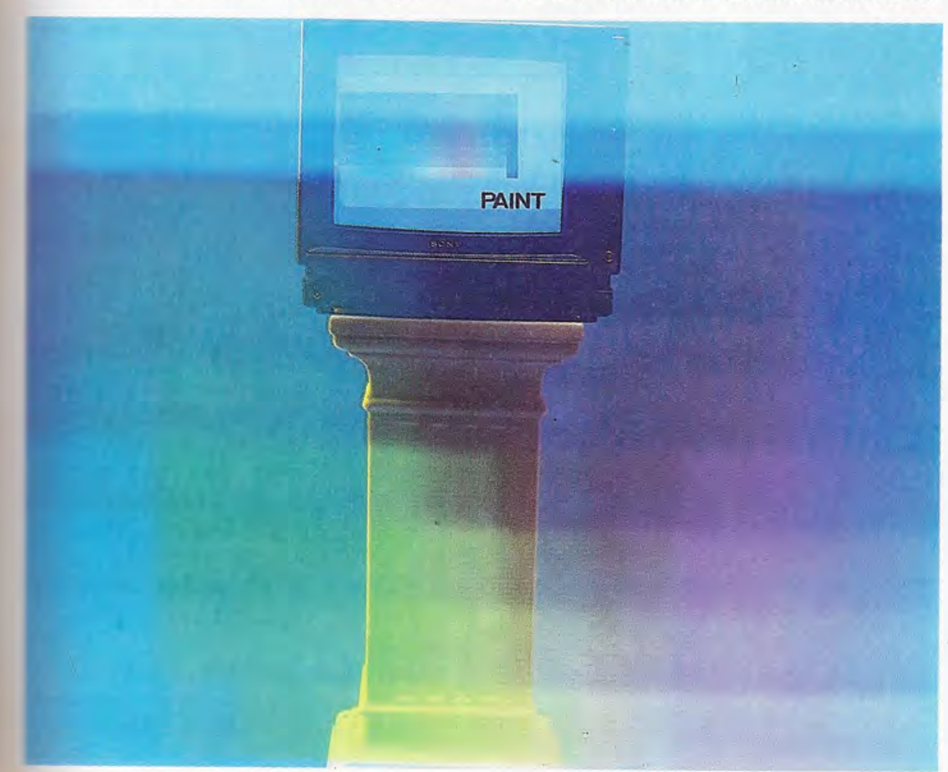
El comando CIRCLE equivale al programa de trazado "manual" de circunferencias mostrado en el capítulo anterior. La circunferencia se va recorriendo en sentido contrario al de las agujas del reloj. El punto inicial es el situado más a la izquierda; esto es, el que se encuentra a la misma altura que el centro. En la circunferencia entera, el punto final coincide con el inicial: se da una vuelta completa volviendo al mismo punto. Esto es lo que se hace en el ejemplo al que nos referimos al variar el contador del bucle de 0 a 6.28 (10 FOR I=0 TO 6.28 STEP 0.01); donde la cantidad 6.28 corresponde a $2*\pi$, que es el número de radianes que contempla la circunferencia. Para trazar arcos con el programa anterior basta con alterar los valores de los límites del comando FOR. De forma análoga, se han de indicar los límites en el trazado de arcos con CIRCLE. Estos dos datos se añaden en el argumento de dicho comando e irán precedidos de un tercero que indica el color del arco (o de la circunferencia completa). A título de ejemplo, la siguiente línea traza una circunferencia con el color número 3.

```
10 CIRCLE (120,75),25,3
```

Por el contrario, la línea que se muestra a continuación sólo presenta la mitad superior de dicha circunferencia.

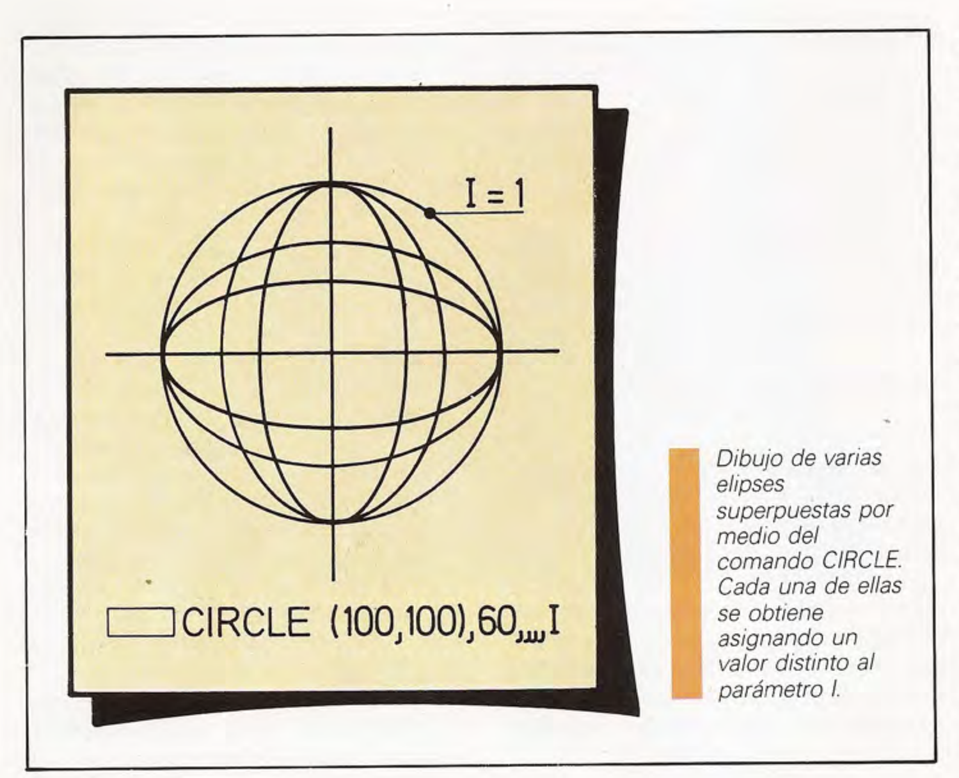
```
10 CIRCLE (120,75),25,3,0,3.14
```

El arco de circunferencia especificado va desde los 0 a los 3.14 radianes (de 0 a π radianes); o lo que es lo mismo, se dibuja tan sólo media circunferencia. Utilizando estos dos nuevos parámetros se pueden construir, pues, toda clase de líneas curvas.



Elipses

Para trazar elipses con la ayuda del comando CIRCLE basta añadir un parámetro más. Este dato se añade al final



Dibujo de varias elipses superpuestas por medio del comando CIRCLE. Cada una de ellas se obtiene asignando un valor distinto al parámetro I.

de la lista situada en el argumento de CIRCLE. El nuevo parámetro proporcionará circunferencias cuando valga 1. Si su valor es menor que 1, el radio especifica la longitud del semieje vertical (esto es, del centro al punto más alto de la elipse). Cuando valga más de 1, el radio indicará el número de pixels que posee el semieje horizontal.

```
100 CIRCLE (100,100),90,3,,,2
```

En esta línea se muestra un ejemplo del trazado de una elipse. Las posiciones empleadas para los puntos inicial y final del arco se han dejado en blanco. Esto significa que se dibujará la elipse entera. Obsérvese el empleo de las comas para no confundir la relación del aspecto con los límites del arco.

El color

Los objetos que se tienen a la vista se identifican por su forma y su color. Los comandos explicados hasta ahora proporcionan formas. Es hora, pues, de entrar en el mundo del color de la mano del BASIC.

La forma de programar los colores a utilizar en el trazado de gráficos es algo muy particular de cada dialecto BASIC. Sin embargo, existen tres zonas específicas de la pantalla en las que se puede emplear el color. Estas corresponden a las siguientes: primer plano (o tinta), fondo (o papel) y borde.

Cuando se escribe texto en la pantalla, las letras son de un color distinto al del fondo. Las letras se consideran trazadas en el color de primer plano. El color de la pantalla vacía es el color de fondo; mientras que el tercer color se refiere a la zona que rodea a la zona utilizable de la pantalla (en aquellos aparatos en los que existe dicha zona).

Existen diferentes métodos para establecer el color que va a tomar cada una de estas zonas. Como ya es habitual, aquí se comentará el método empleado por el BASIC de Microsoft; las diferencias con otros dialectos se muestran en la correspondiente tabla de conversión.

El BASIC de Microsoft utiliza el comando COLOR para especificar el color de cada zona en la pantalla (primer plano, fondo y borde). No obstante, los comandos gráficos llevan todos un parámetro que especifica el color que han de utilizar. Esto permite jugar con varios colores de primer plano. En resumen, el formato general del comando COLOR es el siguiente:

(Número de línea) COLOR [<primer plano>][,[<fondo>][,<borde>]]

en donde la ausencia de un parámetro revela que la zona afectada no se desea modificar. Así, por ejemplo, la siguiente sólo variará el color del borde de la pantalla.

```
120 COLOR,,2
```

El número de colores disponibles es también muy particular de cada máquina. Los diferentes conjuntos de colores pueden variar desde 2 (blanco y negro) hasta 256 o más. Por regla general, cada color utilizable lleva asociado un número o código. De esta forma, cada color se identifica por medio del respectivo código. Este es el método puesto en práctica en los comandos gráficos comentados.

Además de todo esto, en algunos modelos de ordenadores el número de colores utilizables depende del modo gráfico en el que se opere. En cualquier

CIRCLE

Traza circunferencias, arcos de circunferencia y elipses en la pantalla.

Formato: CIRCLE(<X>,<Y>),<rad>,<col>,<ini>,<fin>,<asp>

Ejemplos:
```
10 CIRCLE (100,100),95
50 CIRCLE (90,85),70,,,,2
```

caso, el número de colores y el código de cada uno de ellos debe consultarse en las páginas del manual correspondiente.

Llenando la pantalla de color

El BASIC de Microsoft dispone de un comando cuya función es la de rellenar superficies de color. Este comando es PAINT (en inglés, pintar). El comando PAINT rellena con el color especificado una superficie cerrada de la pantalla. Para su empleo es preciso delimitar previamente la superficie a colorear para que ésta quede totalmente cerrada. Ello se puede conseguir con el uso de LINE y/o CIRCLE.

Una vez dibujado el contorno basta con ejecutar el comando PAINT indicando un punto interior de la superficie. Según la localización del punto especificado, se rellenará una u otra superficie.

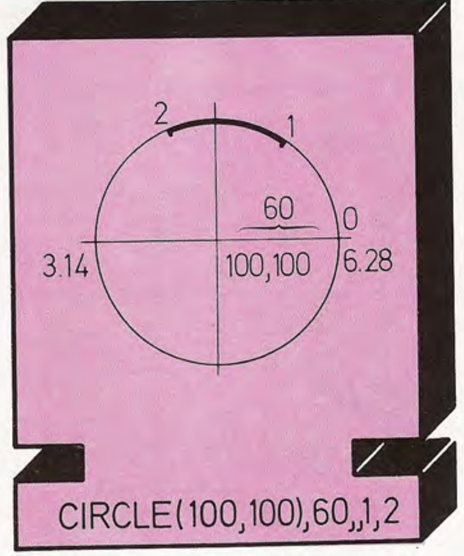
El empleo de un punto perteneciente a la superficie a rellenar es fundamental. Si se traza una circunferencia, se puede pintar tanto su interior como la parte exterior de la misma. Así pues, el punto de referencia identifica unívocamente la zona de la pantalla que se desea colorear. Por ejemplo:

```
30 CIRCLE (100,100),90,2
40 PAINT (100,100),2
█
```

Estas dos líneas trazan una circunferencia y la llenan del color número 2. Se ha empleado el centro de la circunferencia como punto de referencia para PAINT, pero hubiera valido cualquiera de los situados en el interior de la misma; por ejemplo:

```
40 PAINT (100,50),2
```

Si el punto especificado se encuentra en la zona externa, ésa será la parte coloreada. Esto mismo es lo que sucede-



Ejemplo de trazado de un arco de circunferencia por medio del comando CIRCLE.

ría con la ejecución de la siguiente línea:

```
40 PAINT (120,100),2
```

En los tres ejemplos mostrados se observa que el color del "relleno" coincide con el de la circunferencia inicial (concretamente con el número 2). Ello no se debe a una coincidencia. El comando PAINT considera como límites válidos aquellos trazos que son del mismo color que el indicado en su argumento. Si no existen en la pantalla líneas del color especificado se rellenará toda la superficie de la misma.

Según lo dicho, el formato en el que se presenta este comando es el que figura a continuación.

(Número de línea) PAINT (<X>, <Y>)[,<color>]

En donde <X> e <Y> indican las coordenadas del punto de referencia. El parámetro que especifica el color es optativo. Como en el resto de los comandos gráficos, si no se adjunta este parámetro se utilizará el color actual de primer plano.

Como averiguar el color de un punto

Se ha visto la forma de dar color a los puntos de la pantalla. Sin embargo, es posible que en algún momento se necesite conocer el color de un determinado punto. Esto se puede hacer tomando buena nota de los puntos que se colorean. Si se utiliza PAINT para rellenar una superficie compleja, el cálculo de los puntos afectados puede resultar extremadamente arduo. De hecho, la localización exacta de los puntos que componen una simple circunferencia constituye una tarea nada fácil.

Conocer el estado de un punto luminoso resulta útil para localizar trazos en la pantalla. Una vez localizados, éstos pueden ser borrados o emplear el dato



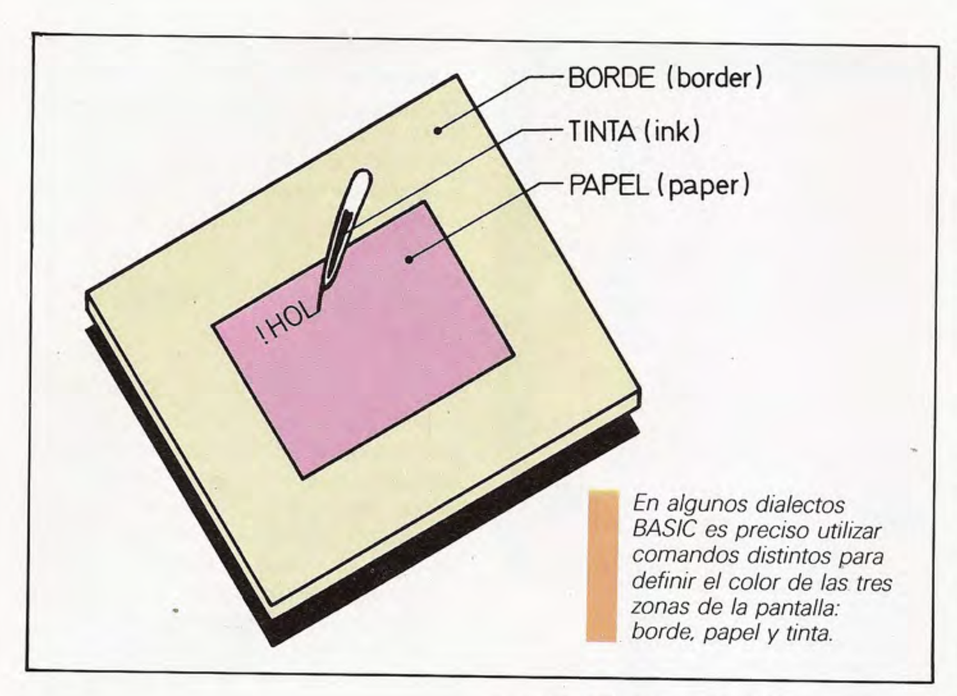
El vocabulario de comandos y funciones del lenguaje BASIC facilita la creación de gráficos en color con la pantalla como lienzo.

POINT

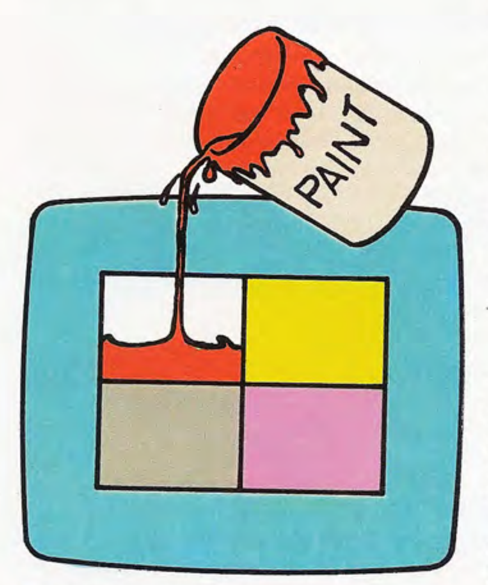
Proporciona el código de color del punto indicado en su argumento

Formato: POINT(<X>,<Y>)

Ejemplos:
```
20 PRINT POINT(125,24)
70 LET COL=POINT(125,24)
```



En algunos dialectos BASIC es preciso utilizar comandos distintos para definir el color de las tres zonas de la pantalla: borde, papel y tinta.



El comando PAINT permite dar color a superficies encerradas por líneas.

que identifica su situación para posteriores cálculos. En el BASIC de Microsoft existe una función que devuelve el código de color de un punto dado. Dicha función es POINT (no confundir con PAINT) y, como es de suponer, admite en su argumento las coordenadas de un punto de la pantalla. Por lo tanto, su formato general es el siguiente:

(Número de línea) [LET] <variable>=POINT(<X>,<Y>)

En el formato se ha incluido POINT en el seno de una sentencia de asignación. Al tratarse de una función, POINT deberá situarse en una expresión numérica o donde se pueda hacer uso del dato que proporciona.

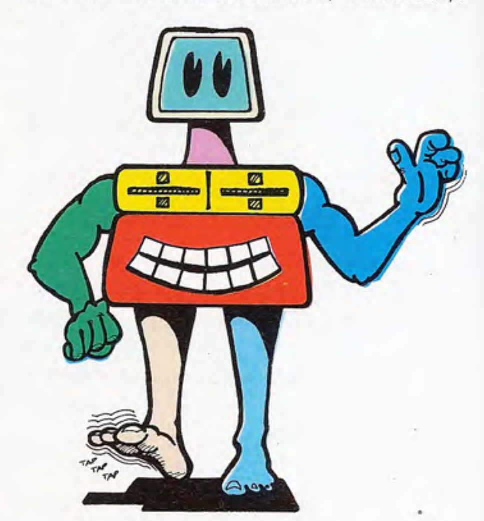
Como ejemplo del uso de POINT se propone un programa que localiza un punto de color 1 en el interior de la línea 100 (horizontal) de la pantalla.

```
300 FOR I=1 TO 256
310 IF POINT(I,100)<>1 THEN NEXT I
320 PRINT I
```

Se ha supuesto que la pantalla contiene 256 puntos horizontales (del 1 al 256). El bucle FOR/NEXT que recorre la línea finaliza cuando se encuentra un punto de color número 1. En ese momento se escribe el valor de I, el cual corresponde a la coordenada X del punto hallado.

Como se ha indicado, POINT proporciona el código de color de un punto. Además de ello, si el punto rebasa de los límites de la pantalla, el valor devuelto será —1. Como los códigos de color son siempre números positivos, el valor negativo indica claramente que se ha inspeccionado un punto que no pertenece a la pantalla. Esta posibilidad puede emplearse para medir, de una forma efectiva, la superficie de la pantalla. El programa que proporciona esta información podría codificarse en BASIC de la siguiente forma.

```
10 REM LIMITES DE LA PANTALLA
20 SCREEN 2
30 X=0
40 X=X+1
50 A=POINT (X,2)
60 IF A <>—1 THEN GOTO 40
70 Y=0
80 Y=Y+1
90 A=POINT (2,Y)
100 IF A <>—1 THEN GOTO 80
110 SCREEN 0
120 PRINT "PIXELS: HORIZONTALES" X; "VERTICALES"; Y
```



Sin lugar a dudas, el color es una de las características más importantes del mundo en que vivimos.

PAINT

Rellena una superficie con el color especificado.

Formato: PAINT (<X>,<Y>),<color>

Ejemplos: 20 PAINT (10,50),3
70 PAINT (100,100)

TABLA DE CONVERSION

Ordenador	Dibujo				Color		
	CIRCLE (X,Y), R,C,I,F,A	Arcos	PAINT (X,Y),C	POINT (X,Y)	TINTA	PAPEL	BORDE
AMSTRAD	—	—	FILL <c>(9)	TEST <x>,<y>	INK <T>,<C>	PAPER <n>	BORDER <n>
APPLE II (APPLESOFT)	—	—	—	—	HCOLOR (7)	—	—
APRICOT (M-BASIC)	—	—	—	—	—	—	—
ATARI	—	—	—	—	COLOR <n>	—	—
CBM 64	—	—	—	—	(8)	(8)	—
DRAGON	CIRCLE (X,Y), R,C,A,I,F	CIRCLE	PAINT (X,Y), C,CC (3)	PPOINT (X,Y)	COLOR T,P		—
EQUIPOS MSX	CIRCLE (X,Y), R,C,I,F,A	CIRCLE	PAINT (X,Y), C,CC (3)	POINT (X,Y)	COLOR T,P,B		
HP-150	—	—	—	—	—	—	—
IBM PC	CIRCLE (X,Y), R,C,I,F,A	CIRCLE	PAINT (X,Y), C,CC (3)	POINT (X,Y)	COLOR T,P,B		
MPF	—	—	—	—	HCOLOR (7)	—	—
NCR DM-V (MS-BASIC)	—	—	—	—	—	—	—
NEW BRAIN	—	—	—	—	—	—	—
ORIC	CIRCLE R,C (1)	—	FILL lin,Col, byte (4)	POINT (X,Y) (5)	INK <n>	PAPER <n>	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	—	COLOR X,Y,T,P		
SINCLAIR QL	CIRCLE X,Y,R	ARC X_1,Y_1 TO X_2, Y_2, ANG	—	—	INK <n>	PAPER <n>	BORDER <n>
SPECTRAVIDEO	CIRCLE (X,Y), R,C,I,F,A	CIRCLE	PAINT (X,Y),C	POINT (X,Y)	COLOR T,P,B		
ZX-SPECTRUM	CIRCLE X,Y,R	DRAW X,Y, ANG (2)	—	POINT (X,Y) (6)	INK <n>	PAPER <n>	BORDER <n>

X,Y: Coordenadas del centro de la circunferencia o del punto a tratar. R: Radio. C: Color. I: Angulo inicial del arco. F: Angulo final del arco. A: Relación de aspecto. ANG: Angulo medido en radianes que corresponde al arco a trazar. T: Código de color de primer plano (tinta). P: Código de color de fondo (papel). B: Código de color del borde.

(1) El color se indica como 0 si es el de fondo ó 1 si es el de primer plano. (2) Actúa como DRAW (ver primer capítulo de gráficos) pero trazando una línea curva. La curvatura viene impuesta por el ángulo en radianes a trazar. (3) CC especifica el color del contorno a rellenar. (4) Sirve para rellenar las líneas y columnas de la pantalla indicadas con el byte que se adjunta. (5) Devuelve el valor −1 si es color de primer plano y 0 si es fondo. (6) Proporciona el valor 0 si corresponde a «papel» o si corresponde a tinta. (7) Es una variable que se actualiza con el código de color en una sentencia de asignación. (8) El color se cambia mediante códigos de control en el argumento de PRINT. (9) Sólo en los CPC664 y CPC6128.

En este ejemplo se crean dos bucles que exploran una línea vertical y otra horizontal. Al alcanzar ambos límites de la pantalla finalizan los bucles. La variable que sirvió de contador dará la medida en pixels de cada lado de la pantalla. La línea 110 se utilizará en aquellos aparatos que no permitan escribir texto en la pantalla de gráficos (SCREEN 2). En el ejemplo se ha consideradoo que el modo de texto se obtiene por medio del comando SCREEN 0.



Variables de tiempo

En un capítulo precedente se habló de las variables del sistema, o lo que es lo mismo: de posiciones de memoria reservadas para uso del sistema. Entre ellas se encuentra una variable que puede resultar de utilidad en múltiples casos.
Se trata de una variable cuyo contenido se incrementa automáticamente mientras el aparato está encendido. Por lo tanto, dicha variable puede ser empleada para medir el tiempo transcurrido entre dos acontecimientos. Muchos dialectos BASIC permiten el acceso directo a esta variable, la cual suele estar asociada a la palabra clave TIME o TIME$ (en el primer caso se trata de una variable numérica).
Dependiendo del ordenador, esta variable se incrementará con mayor o menor frecuencia; por ejemplo, es habitual que se incremente en una unidad cada sesentavo de segundo.
La referida variable permite medir el tiempo de ejecución de un programa. Por ejemplo, el siguiente programa calcula el tiempo invertido por el ordenador en ejecutar 10.000 sumas:

```
10 LET T=TIME
20 FOR I=1 TO 10000
30 LET A=B+C
40 NEXT I
50 PRINT 60*(TIME—T)
```

El valor inicial de TIME es almacenado en la variable T. Una vez realizado el proceso, se calcula el tiempo transcurrido por medio de una simple resta (TIME—T). Si el ordenador cuenta en sesentavos de segundo, habrá que multiplicar la cantidad obtenida por 60. Este es, precisamente, el cometido de la línea 50.
Algunos ordenadores actualizarán el valor de TIME cada segundo, lo cual significa que la lectura de esta variable vendrá directamente en segundos. En tal caso, la multiplicación por sesenta es superflua.
Cuando la variable de tiempo no es TIME sino TIME$, la cosa cambia. El carácter "$" final indica que se trata de una variable de cadena. Esta circunstancia impide su tratamiento aritmético. Además, TIME$ suele emplear un formato del tipo *hh:mm:ss*. Esto se verá más claro con un ejemplo:

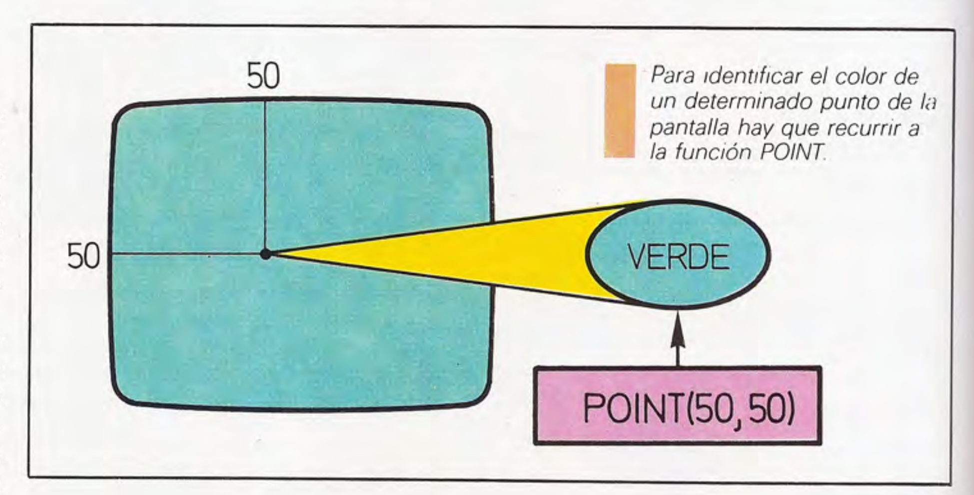
```
PRINT TIME$<CR
0:23:35
■
```

Dicha variable puede ser actualizada mediante una sentencia de asignación del tipo LET TIME$="12.25:30". Otros aparatos disponen además de una variable que almacena la fecha. Esta última suele estar asociada a la palabra DATE$.

Color y texto

Se ha indicado más arriba que el comando que controla el color es precisamente COLOR. Este comando establece los colores de primer término, fondo y borde. Su acción afecta tanto a gráficos como a texto. En el caso del texto los caracteres se escriben en el color de primer plano.

Un efecto parcial del cambio de color es el que no altera el texto escrito con anterioridad. Esto quiere decir que sólo se cambiará el color del texto que se escriba a continuación. Esta característica permite obtener zonas de la pantalla con distinto color de fondo o de primer término (o ambos). Así se consigue destacar uno o varios fragmentos de texto.



Para identificar el color de un determinado punto de la pantalla hay que recurrir a la función POINT.

Otras posibilidades utilizables en la representación de texto son el vídeo inverso y el parpadeo. El primero se refiere a la permutación del color de papel de la tinta. Ello significa que el carácer se representará en el color del anterior fondo y el fondo en el del anterior primer término; esto es, invertidos. El otro efecto, el de parpadeo, produce el paso repetido de vídeo normal a vídeo inverso, lo que implica una serie de destellos que se repetirán con una frecuencia fija.

Los comandos que controlan estos dos efectos, suelen situarse en el argumento de PRINT; habitualmente se formulan con las palabras INVERSE y FLASH (o similares).

COLOR

Establece el color de primer plano, fondo y borde de la pantalla:

Formato: COLOR <primer plano>,<fondo>,<borde>

Ejemplos: 20 COLOR 1,2,2
70 COLOR 4, 15, 5

Macrolenguajes gráficos

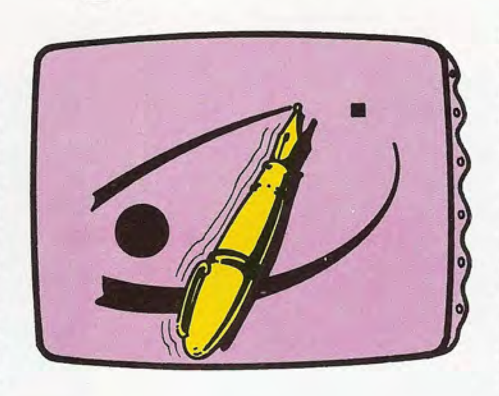
En la exposición hecha hasta ahora sobre los gráficos se ha hablado de los comandos más habituales en BASIC. Sin embargo, en las tablas de compatibilización de los capítulos precedentes se observan importantes huecos. Ello no significa que los correspondientes aparatos carezcan de instrucciones para el manejo de gráficos. Lo que ocurre es que utilizan otros métodos para ese tratamiento.

Un macrolenguaje viene a ser un sublenguaje de comandos asociado a una palabra clave del BASIC. Los macrolenguajes gráficos suelen emplear comandos para el desplazamiento del cursor de gráficos. Con esos desplazamientos se consigue definir el dibujo, por el método de recorrerlo. Algo muy similar a la técnica empleada en el denominado Turtle Graphics del Logo (véanse los capítulos dedicados a este lenguaje).

De los diferentes macrolenguajes gráficos, se comenta en estas páginas el propio del dialecto BASIC de Microsoft, por ser el más ampliamente utilizado.

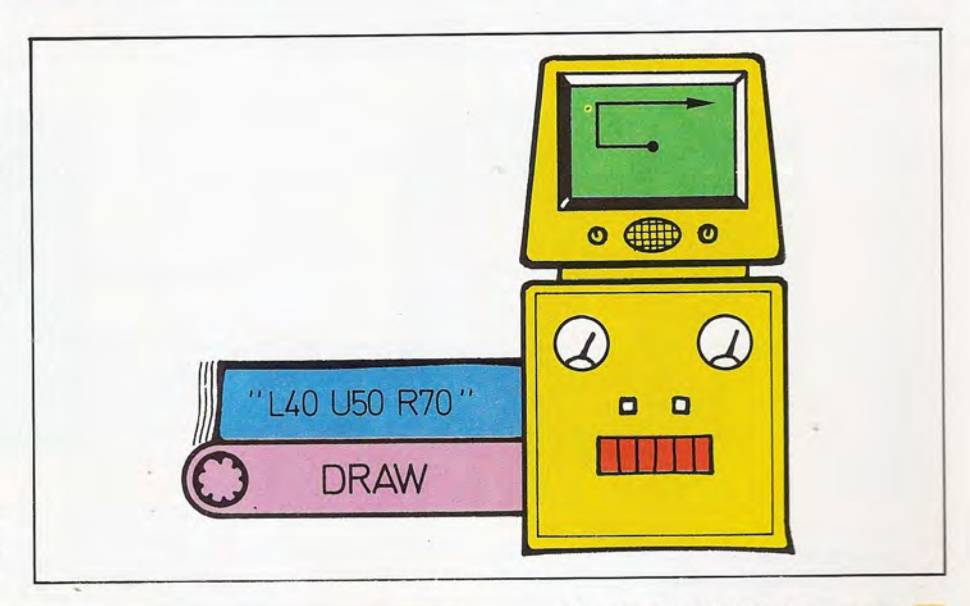
La filosofía del macrolenguaje

En el BASIC de Microsoft se hace uso de un macrolenguaje gráfico asociado a la palabra DRAW. Este comando es el que nos introduce en el nuevo lenguaje de comandos. En cierto modo guarda una ligera relación con el otro comando DRAW, el comentado en el primer capítulo de gráficos. Aquél trazaba una recta desde el último punto referenciado hasta el indicado mediante sus coordenadas relativas. Es decir, una vez situadas en un punto se saltaba a otro indicando el incremento que debían sufrir ambas coordenadas. Para ello se empleaban dos enteros que especificaban dicho incremento. El nuevo comando DRAW también toma como base el último punto trazado. En su argumento se indica la lista de instrucciones del macrolenguaje que se desean ejecutar.



El macrolenguaje gráfico: una forma de hablar con el ordenador de temas pictóricos.

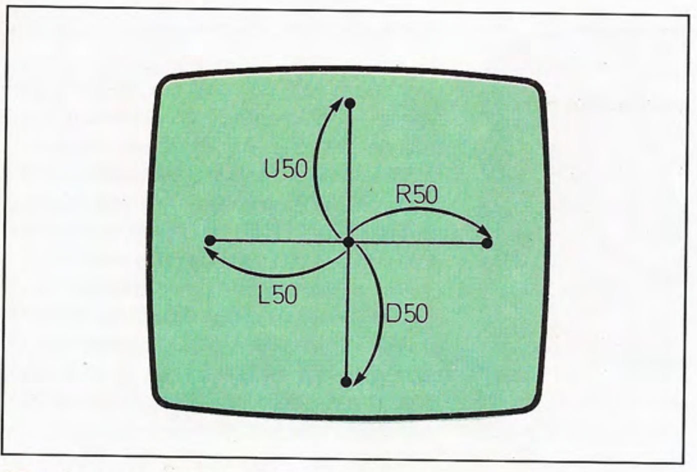
Otras formas de manejar los gráficos



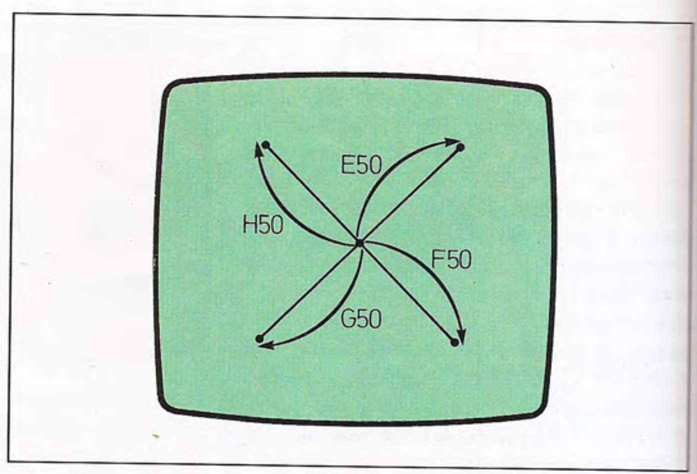
El comando DRAW da paso a la introducción de una cadena ejecutable que el ordenador convierte en gráficos.



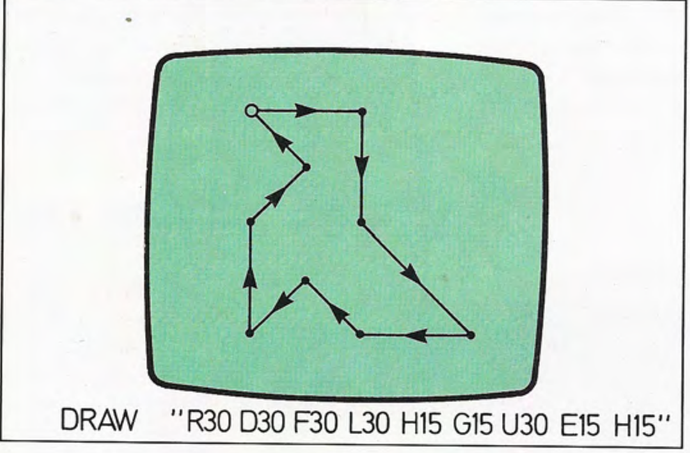
Los ordenadores adscritos a la norma MSX poseen el macrolenguaje gráfico asociado al comando DRAW dentro de su intérprete BASIC.



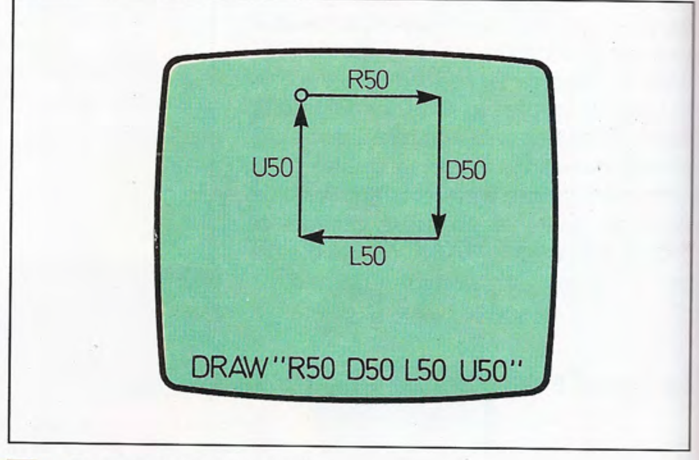
Los cuatro comandos principales del macrolenguaje gráfico permiten el movimiento vertical y horizontal.



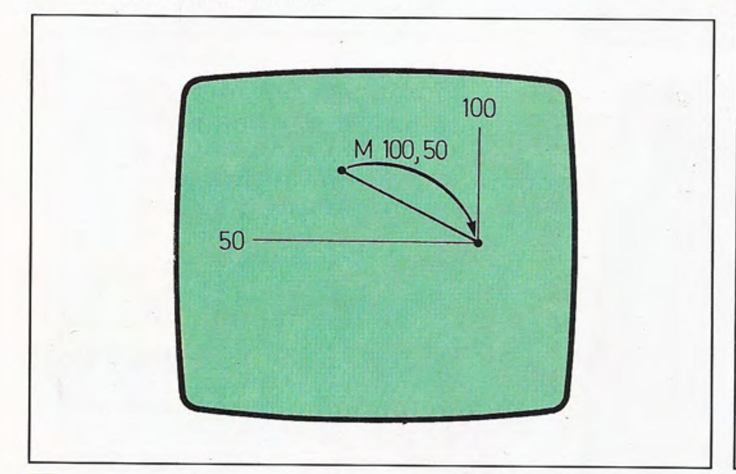
El trazado de líneas inclinadas se encomienda a los subcomandos E, F, G y H.



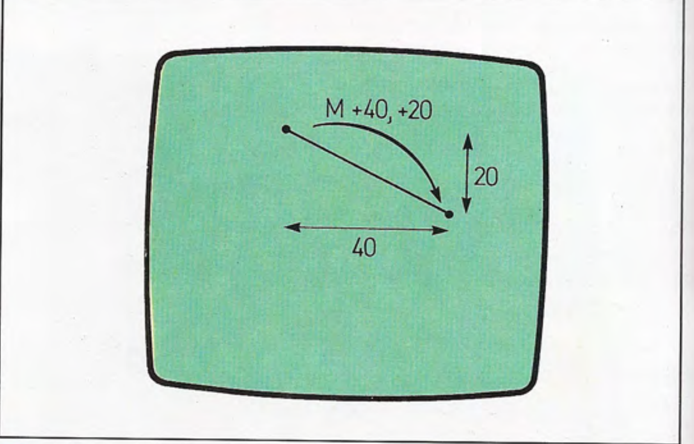
Realización de un dibujo utilizando los subcomandos de desplazamiento en las ocho direcciones.



Trazado de un cuadrado mediante una sencilla cadena de instrucciones.



Los desplazamientos a un punto dado por sus coordenadas absolutas se realizan haciendo uso del comando M.



El comando M también permite el desplazamiento a un punto dado por sus coordenadas relativas al último punto trazado.

El primer paso

Ya se ha comentado que el comando DRAW utiliza como punto de partida el último punto trazado. Para iniciar un dibujo se puede marcar ese punto inicial por medio del comando PSET.

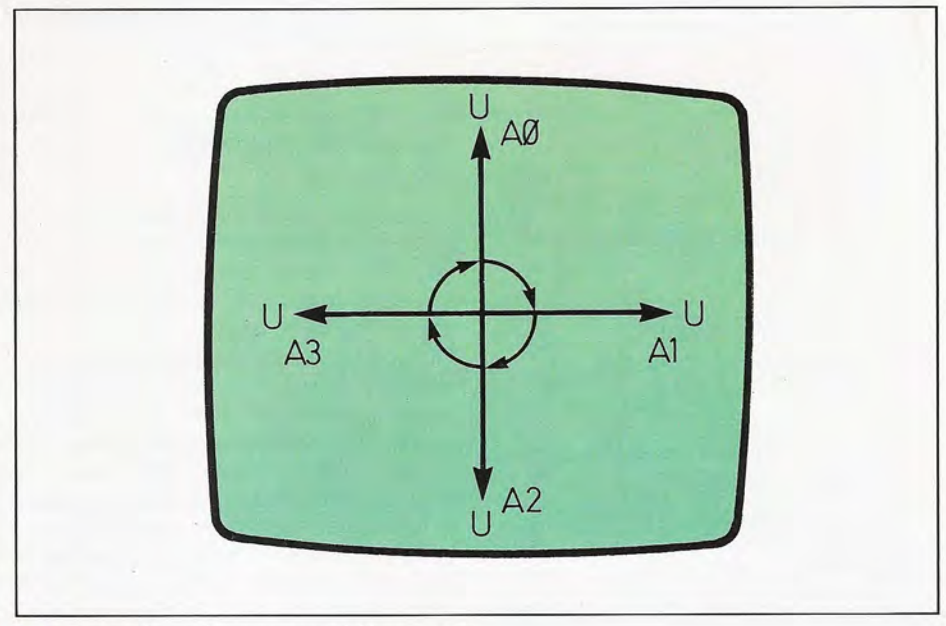
Una vez posicionado el cursor de gráficos se trazarán líneas con DRAW. Para ello se emplean cuatro letras del macrolenguaje: U(up) arriba, D(down) abajo, L(left) izquierda y R(right) derecha. Estas son las cuatro direcciones principales de desplazamiento del macrolenguaje. A continuación de cada comando-letra se ha de especificar la magnitud del movimiento. Dicha magnitud se mide en pixels y será, por lo tanto, un número entero. He aquí el primer programa que utiliza este macrolenguaje:

```
10 SCREEN 2
20 PSET (100,100)
30 DRAW "R50D50L50U50"
40 GOTO 40
█
```

En este ejemplo, tras fijar el punto 100,100, se ha desplazado el cursor 50 pixels a la derecha, 50 abajo, 50 a la izquierda y 50 arriba. Todo ello completa un cuadrado de lado 50 cuyo vértice superior derecho se encuentra en el punto 100,100. Como puede apreciarse en el listado, las cuatro órdenes de la cadena ejecutable no se separan entre sí, van contiguas.

Este programa podría complicarse un poco más empleando los comandos de tratamiento de cadenas. El siguiente ejemplo permite trazar un cuadrado de lado variable:

```
5 INPUT A
10 SCREEN 2
20 PSET (100,1000)
30 A$=STR$(A)
40 C$="R"+A$+"D"+A$'D"+A$+"L"+A$+"U"+A$
50 DRAW C$
60 GOTO 60
```



Dirección en la que actuará U con cada una de las cuatro orientaciones posibles.

Trazando diagonales

Además de las direcciones horizontal y vertical se pueden emplear las que forman un ángulo de 45 grados con éstas. Las nuevas direcciones funcionan de la misma forma que las anteriores. Sus letras identificativas son: E arriba a la derecha, F abajo a la derecha, G abajo a la izquierda y H arriba a la izquierda. Si las anteriores se identificaban por coincidir con las iniciales de la palabra inglesa, éstas se pueden recordar por marcar su orden el movimiento de las agujas del reloj. Empleando estas nuevas instrucciones se puede trazar un cuadrado apoyado sobre un vértice:

```
10 SCREEN 2
20 PSET (100,100)
30 DRAW "E50F50G50H50"
40 GOTO 40
```

También se puede hacer uso del manejo de cadenas para determinar la longitud del lado.

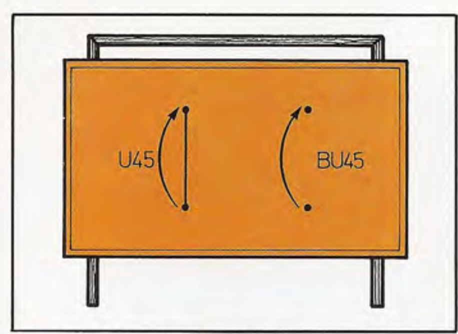
```
5 INPUT A
10 SCREEN 2
20 PSET (100,100)
30 A$=STR$(A)
40 C$="E"+A$+"F"+A$+"F"+A$+"D+A$+"H"+A$
50 DRAW C$
60 GOTO 60
```

Teniendo acceso a esas ocho direcciones se pueden crear figuras relativamente complejas. Como ejemplo se muestra un pequeño programa que traza el contorno de una pajarita de papel:

```
10 SCREEN 2
20 PSET (100,100)
30 DRAW "R30D30F30L30H15G15U30E15H15"
40 GOTO 40
```

Movimiento hacia un punto absoluto

El movimiento visto hasta ahora permite ocho direcciones de desplazamiento. En la especificación de dicho movimiento se incluye el salto en pixels. Ello implica un movimiento relativo al último punto trazado. También puede indicarse el desplazamiento por las coordenadas absolutas del punto final del trazo. Ello se consigue con la letra M (de mover), a la que habrán de adjuntarse dos datos: los correspondientes a las coordenadas absolutas del punto en cuestión. El siguiente ejemplo traza el



El prefijo B permite el desplazamiento sin trazar la trayectoria.

mismo cuadrado del principio utilizando esta nueva posibilidad:

```
10 SCREEN 2
20 PSET (100,100)
30 DRAW "M150,100M150,150M100,
   150M100,100"
   40 GOTO 40
```

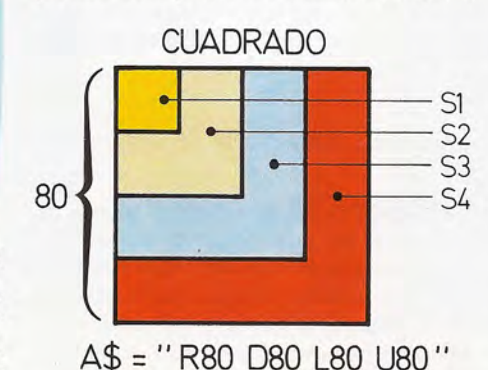
En este caso se ha supuesto que el origen de coordenadas se encuentra, como es habitual, en la esquina superior izquierda de la pantalla. Si no es así, se trazará el cuadrado hacia arriba, en lugar de hacia abajo.

En el ejemplo se pone de manifiesto la formulación de esta nueva posibilidad. Tras la letra M se añaden ambas coordenadas (horizontal y vertical) separadas por una coma.

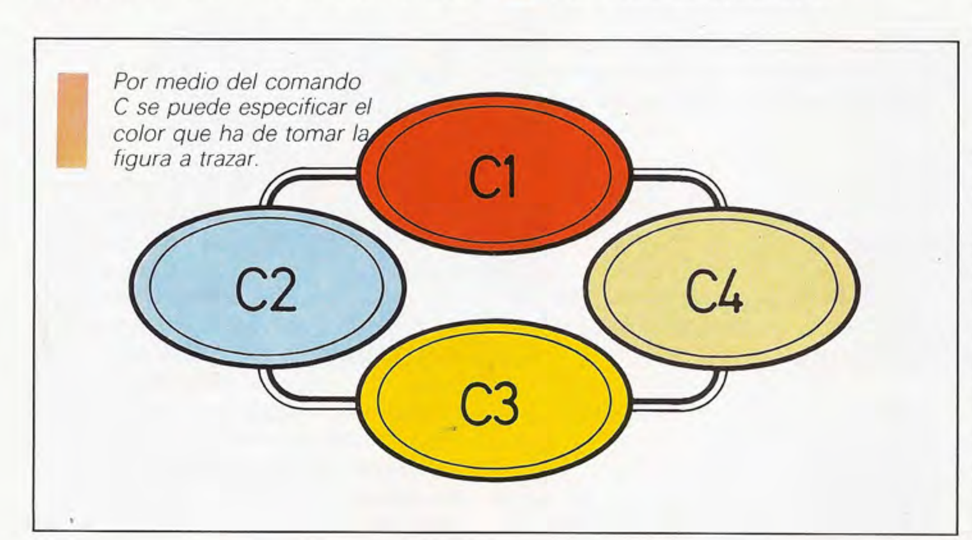
Caminando sin dibujar

Tal como se ha visto hasta ahora, se pueden realizar figuras conexas; esto es, de un trazo continuo. La forma de cambiar de sitio el cursor sin por ello trazar una línea parece relegada al empleo de PSET. No obstante, puede prescindirse de ese comando. Ello resultará útil a la hora de encadenar varias figuras en una sola cadena ejecutable.

La letra B colocada delante de un comando-letra de movimiento producirá un desplazamiento sin trazo. Esta letra es la inicial de «blank», que en inglés significa «en blanco». Así pues, no es preciso anteponer un comando PSET a



Una misma figura se puede representar en distintos tamaños; para ello basta con cambiar la escala por medio del comando S.



Por medio del comando C se puede especificar el color que ha de tomar la figura a trazar.

la ejecución de una cadena gráfica. El ejemplo inicial del cuadrado puede resumirse en una sola cadena.

```
10 SCREEN 2
20 DRAW "BM100,100R50D50L50U50"
30 GOTO 30
```

Como se aprecia en el ejemplo propuesto, el prefijo B sólo afecta a la siguiente instrucción, y no necesita de más parámetros. Este prefijo puede ser empleado con el resto de comandos-letra, permitiendo el trazado de varias figuras separadas.

Rotación

La mayor parte de las veces se desea utilizar una figura repetidamente. Para ello se puede almacenar una cadena generadora de dicha figura en una variable. Si la cadena utiliza comandos relativos, ello permitirá situar la figura en distintas posiciones de la pantalla. Véase un ejemplo en el que se dibuja el mismo cuadrado en diferentes lugares

```
 10 SCREEN 2
 20 A$="R40D40L40U40"
 30 DRAW "BM20,20"
 40 DRAW A$
 50 DRAW "BM20,100"
 60 DRAW A$
 70 DRAW "BM100,20"
 80 DRAW A$
 90 DRAW "BM100,100"
100 DRAW A$
200 GOTO 200
```

Pero, una vez creada la cadena, ésta sólo puede ser reubicada en otro lugar. Para alterar su forma se precisa un tratamiento de la propia cadena. Esto no es siempre cierto. Por ejemplo, se puede especificar una rotación de la figura creada. Para ello se hace uso de una nueva letra-comando. Se trata de la letra A (de Angulo), que seguida de un número varía la orientación de las instrucciones que le sigan.

El comando A admite como argumento un entero del 0 al 3. El cero indica la orientación inicial: el comando U moverá hacia arriba. El resto de números marca un ángulo de desfase de 90, 180

TABLA DE CONVERSION							
Ordenador	Comando BASIC	Subcomandos del macrolenguaje gráfico					
	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
AMSTRAD	DRAW	—	—	—	—	—	—
APPLE II (APPLESOFT)	—	—	—	—	—	—	—
APRICOT (M-BASIC)	—	—	—	—	—	—	—
ATARI	—	—	—	—	—	—	—
CBN 64	—	—	—	—	—	—	—
DRAGON	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
EQUIPOS MSX	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
HP-150	—	—	—	—	—	—	—
IBN PC	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
MPF	—	—	—	—	—	—	—
NCR DM-V (MS-BASIC)	—	—	—	—	—	—	—
NEW BRAIN	—	—	—	—	—	—	—
ORIC	—	—	—	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	—	—	—	—
SINCLAIR QL	—	—	—	—	—	—	—
SPECTRAVIDEO	DRAW	U,R,D,L	E,F,G,H	M<h>,<v>	M±<h>,±<v>	B,N	A,S,C
ZX-SPECTRUM	—	—	—	—	—	—	—

y 270 grados respectivamente. Ello quiere decir que tras la ejecución de A1, el comando U efectuará un movimiento hacia la derecha de la pantalla, con A2U moverá hacia abajo, etc. El resto de comandos relativos variará su dirección de ataque solidariamente con el comando U.

Esta nueva función gráfica permite repetir una misma figura con cuatro orientaciones distintas. Esto mismo es lo que se realiza en el siguiente programa:

```
10 SCREEN 2
20 A$="R40D40L40U40"
30 DRAW "BM100,100"
40 DRAW A$
50 DRAW "A1"
60 DRAW A$
70 DRAW "A2"
80 DRAW A$
90 DRAW "A3"
100 DRAW A$
200 GOTO 200
```

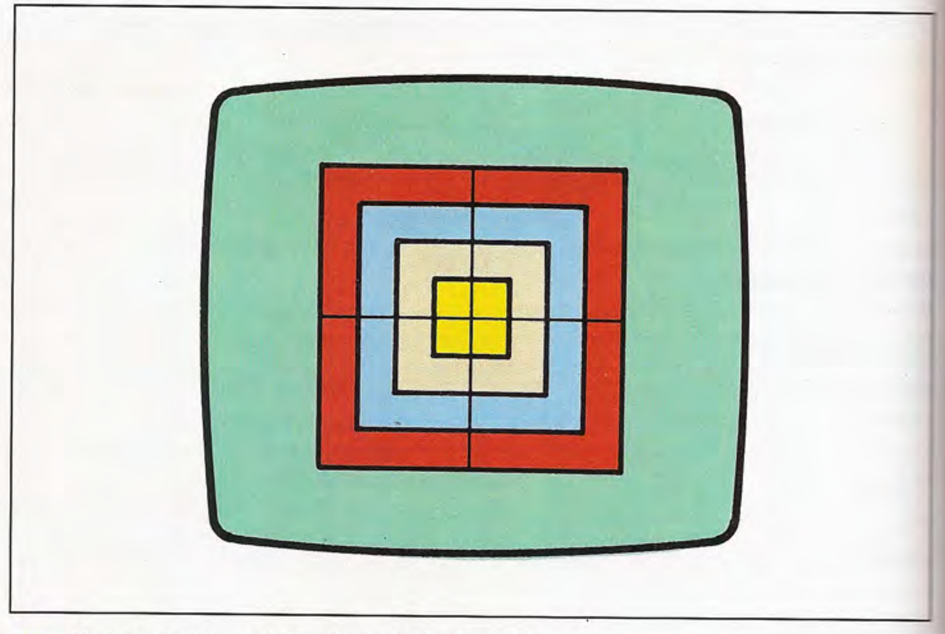
Escala

El cambio de escala se efectúa mediante el nuevo comando-letra S (de Scale). La nueva escala se indica con un entero tras la letra S. El entero puede variar de 1 a 255, siendo la escala habitual la que corresponde a S4. En realidad, el factor de escala se obtiene de la división del entero propuesto entre 4.

El dato así obtenido es el que se multiplicará a las longitudes de trazo especificadas. De esta manera, al dividir 4 entre 4 da como resultado 1, con lo que la escala es la natural (medida en pixels). Ello permite reducir la figura hasta cuatro veces (S1 de un factor de 1/4) o ampliarle hasta 63 veces (252/4=63).

A continuación se muestra un ejemplo en el que se trazan cuatro cuadrados, cada uno en una escala diferente. El resultado es el de cuatro cuadrados uno dentro de otro.

```
10 SCREEN 2
20 A$="R40D40L40U40"
30 DRAW "BM100,100"
40 DRAW A$
50 DRAW "S1"
60 DRAW A$
70 DRAW "S2"
80 DRAW A$
90 DRAW "S3"
100 DRAW A$
200 GOTO 200
```



¡Y ahora todo junto! Resultado de la ejecución del último ejemplo del texto.

Al igual que el comando A, el de cambio de escala S queda fijado hasta que se vuelve a indicar una nueva escala. Su acción sólo afecta a los comandos de desplazamiento relativo, ya que las coordenadas no son susceptibles de alteración.

Color

Las líneas trazadas hasta el momento adoptan el color del primer plano. Color que habrá sido estipulado mediante la ejecución del apropiado comando COLOR (véase el capítulo precedente de gráficos en color). Si el aparato dispone de más colores, éstos se podrán elegir en el seno de una cadena ejecutable asociada a DRAW. Para ello se emplea un nuevo comando-letra. Se trata en esta ocasión de la letra C (de Color), la cual deberá ir seguida del código correspondiente al pigmento deseado.

Subcomandos del macrolenguaje gráfico del Microsoft	
U<n>	Mueve el cursor de gráficos n pixels hacia arriba
D<n>	Mueve el cursor de gráficos n pixels hacia abajo
R<n>	Mueve el cursor de gráficos n pixels hacia la derecha
L<n>	Mueve el cursor de gráficos n pixels hacia la izquierda
E<n>	Mueve el cursor en la diagonal arriba-derecha
F<n>	Mueve el cursor en la diagonal abajo-derecha
G<n>	Mueve el cursor en la diagonal abajo-izquierda
H<n>	Mueve el cursor en la diagonal arriba-izquierda
M<h>,<v>	Mueve el cursor al punto de coordenadas absolutas <h> y <v>
M±<h>,±<v>	Mueve el cursor al punto de coordenadas relativas <h> y <v>
B	Prefijo que evita que se marque la trayectoria del cursor
N	Prefijo que, tras trazar un segmento, deja el cursor en el punto de partida
A<n>	Impone una rotación a la figura que se plasme a continuación
S<n>	Permite dibujar una figura a escalas diferentes
C<n>	Indica el color que ostentarán los siguientes trazos en la pantalla

El siguiente ejemplo traza cuatro cuadrados, cada uno en un color diferente:

```
10 SCREEN 2
20 A$="R40D40L40U40"
30 DRAW "BM20,20"
35 DRAW "C0"
40 DRAW A$
50 DRAW "BM20,100"
55 DRAW "C1"
60 DRAW A$
70 DRAW "BM100,20"
75 DRAW "C2"
80 DRAW A$
90 DRAW "BM100.100"
95 DRAW "C3"
100 DRAW A$
200 GOTO 200
```

Los códigos correspondientes a cada color dependerán del aparato en particular.

A la hora de elegir el color es preciso tener en cuenta el color de fondo. Si se dibuja un figura en el mismo color que el fondo, ésta no será visible.

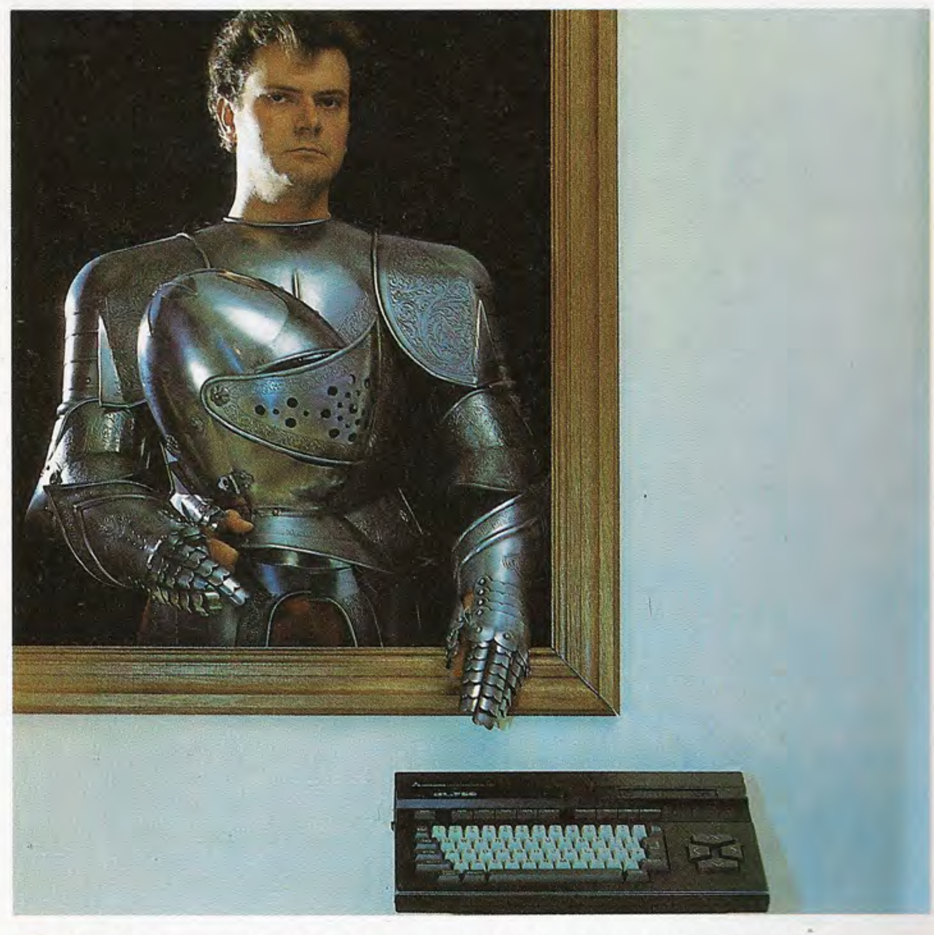
Un vez cambiado el color del trazo, este permanecerá como color en curso hasta que vuelva a cambiar. Para volver al color de primer término inicial puede emplearse tanto la opción C como el comando COLOR.

Otras herramientas gráficas

Caracteres programables y «sprites»



Entre las posibilidades gráficas comentadas hasta el momento se encuentran el trazado de rectas y curvas y el manejo de los colores. Todo ello se realiza por medio de los comandos específicos de que está dotado el ordenador. Otro método para crear dibujos reside en el uso de los denominados caracteres semigráficos. Estos caracteres especiales permiten plasmar figuras predefinidas a través de un comando PRINT. Los caracteres semigráficos suelen incluir pequeños redondeles y cuadrados del tamaño de un carácter. En el presente capítulo se aborda la redefinición de caracteres y otras herramientas que permiten un uso más completo de las posibilidades de la pantalla.



Las técnicas para la definición y tratamiento de sprites permiten crear formas gráficas y controlar su «animación» sobre la pantalla conectada al ordenador.

Los caracteres

Los caracteres que presenta el ordenador en la pantalla están formados por puntos. La disposición de estos puntos da lugar a la «matriz» del carácter. Por regla general, la matriz consiste en un cuadrado de ocho por ocho puntos. Cada punto puede estar encendido o apagado, dando lugar así al carácter a representar. Por ejemplo, el carácter «A» puede representarse mediante la siguiente matriz:



```
. . . ● . . . .
. . ● . ● . . .
. ● . . . ● . .
● . . . . . ● .
● ● ● ● ● ● ● .
● . . . . . ● .
● . . . . . ● .
. . . . . . . .
```

En donde el símbolo «.» significa apagado y el «●» encendido. Es norma usual dejar libres la última fila y columna para que se cree una separación entre caracteres contiguos. Como se puede apreciar, cada línea tiene ocho puntos. Esto quiere decir que cada una de las filas que forman el carácter puede almacenarse en un solo byte. Por lo tanto, para almacenar la imagen de un carácter se necesitan ocho bytes.

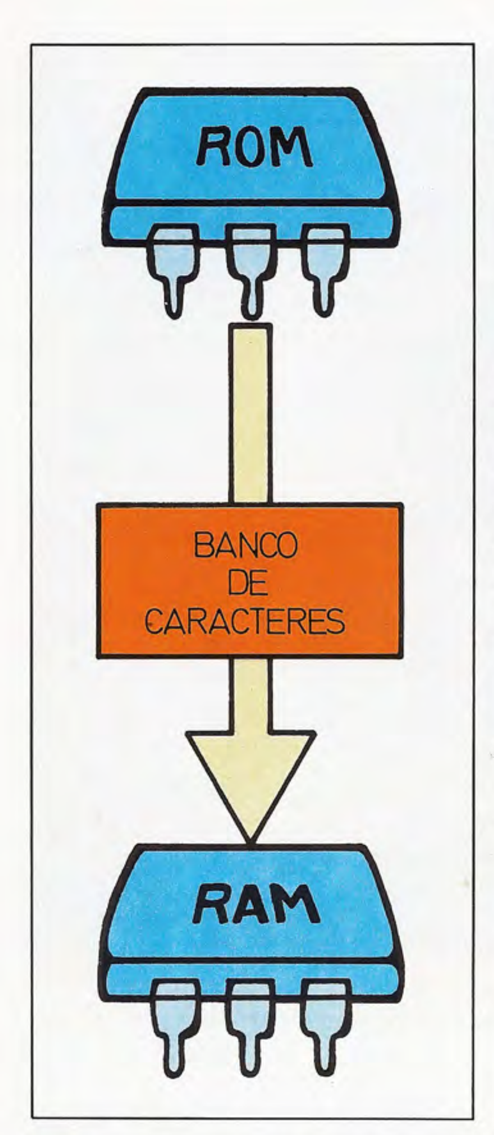
El ordenador no conoce de por sí la imagen de cada carácter, y por ello ha de tener una zona de memoria dedicada al almacenamiento de dichas imágenes. Esta zona suele estar localizada en la memoria ROM (de sólo lectura) y se denomina «banco de caracteres». Cuando es necesario representar un carácter en pantalla, el ordenador busca la imagen en el banco de caracteres y la proyecta en la posición adecuada de la pantalla.

De lo señalado se deduce que para crear un nuevo juego de caracteres bastaría con variar el contenido de los bytes del banco de caracteres. Desafortunadamente esto no es posible, ya que dicho banco se encuentra almacenado en ROM (la zona de memoria cuyo contenido no es posible alterar). De todas formas existen «trucos» para solventar ese problema.

Programando caracteres

Supongamos por un momento que el banco de caracteres se encontrara en memoria RAM (memoria cuyo contenido sí puede ser alterado). En tal caso, para cambiar un carácter bastaría con «encender» o «apagar» los bits oportunos de las posiciones de memoria en las que se encuentra almacenado el carácter en cuestión.

El comando BASIC que permite el acceso a posiciones individuales de memoria es POKE, comando que es explicado en profundidad en el capítulo «Del BASIC al código máquina» de este mis-



Para que sea posible alterar el repertorio de caracteres, éste ha de ser previamente trasladado de memoria ROM a RAM.

mo tomo. El uso de POKE no facilita el acceso a cada bit individualmente, sino al byte en conjunto. Ello obliga al cambio de una fila completa de la matriz del carácter.

El siguiente paso consiste en identificar el efecto que produce el almacenamiento de un entero en una de estas posiciones de memoria. La cosa no puede ser más sencilla: la sucesión de ceros y unos (bits apagados o encendidos) de cada fila se corresponde con el entero introducido, pero escrito en el sistema binario. Por ejemplo, el almacenamiento del número 9 en una de esas posiciones produciría una línea de este tipo:

```
....●..●
```

Ello se debe a que la configuración binaria 00001001 (o más sencillamente 1001) corresponde al número decimal 9. Así pues, conociendo el paso de binario a decimal se puede alterar la imagen de un carácter sin más que ejecutar los adecuados POKES.

Para aclarar algo más las ideas se va a mostrar un ejemplo de lo anteriormente mencionado. Se parte de la base de que los caracteres se encuentran en RAM. El carácter a alterar es el «A», que por comodidad supondremos situado en la posición número 1000. Las posiciones de memoria que interesan son las siguientes:

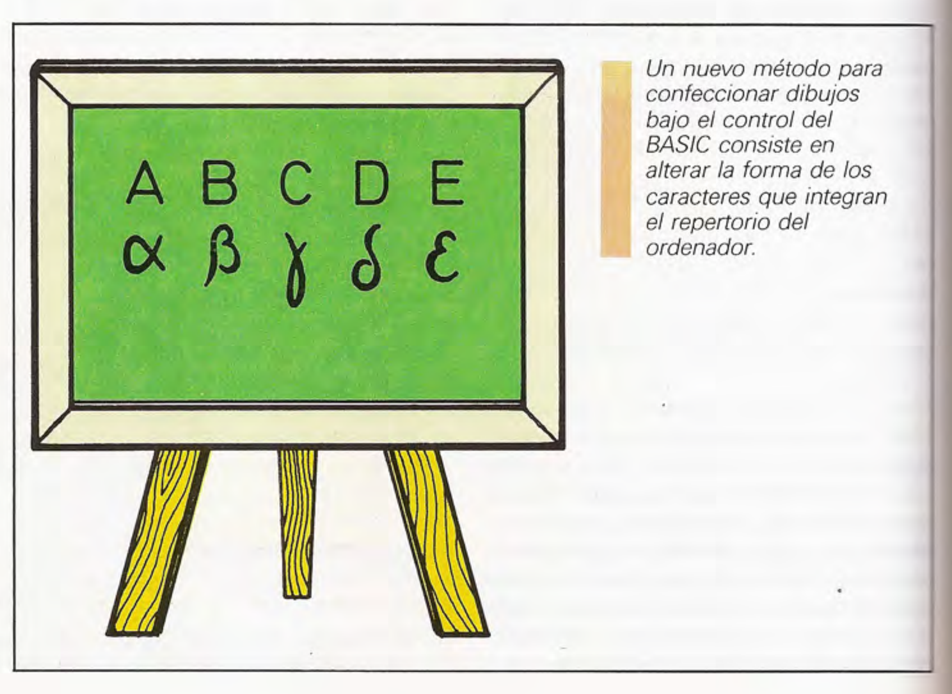
```
1000 ...●....
1001 ..●.●...
1002 .●...●..
1003 ●.....●.
1004 ●●●●●●●.
1005 ●.....●.
1006 ●.....●.
1007 ........
```

Tras el cambio, se desea que el carácter adquiera el siguiente aspecto:

```
1000 .●●●●●..
1001 ●.....●.
1002 ●.....●.
1003 ●.....●.
1004 ●●●●●●●.
1005 ●.....●.
1006 ●.....●.
1007 ........
```

Es decir, sólo se alterarán las tres primeras filas. El paso siguiente consiste en calcular los números decimales que corresponden a los binarios 01111100 y 10000010. Para ello emplearemos una sencilla regla: a cada «uno» que aparece en el número binario se le asigna una cantidad dependiendo de la posición en la que se encuentra; luego, basta con sumar esas cantidades para obtener el valor decimal. Las cantidades claves coinciden con las ocho primeras potencias de 2. Contando de derecha a izquierda, los respectivos unos valen 1, 2, 4, 8, 16, 32, 64 y 128. De esta forma, el número 01111100 vale 4+8+16+32+64=124 y el 10000010 vale 2+128=130: los valores decimales 124 y 130, respectivamente.

Tras este sencillo cálculo sólo queda por almacenar los datos calculados en las posiciones adecuadas. Esto último se realiza por medio del comando POKE. Estas serían las instrucciones oportunas:



Un nuevo método para confeccionar dibujos bajo el control del BASIC consiste en alterar la forma de los caracteres que integran el repertorio del ordenador.

```
POKE 1000,124
POKE 1001,130
POKE 1002,130
```

Reubicación del banco de caracteres

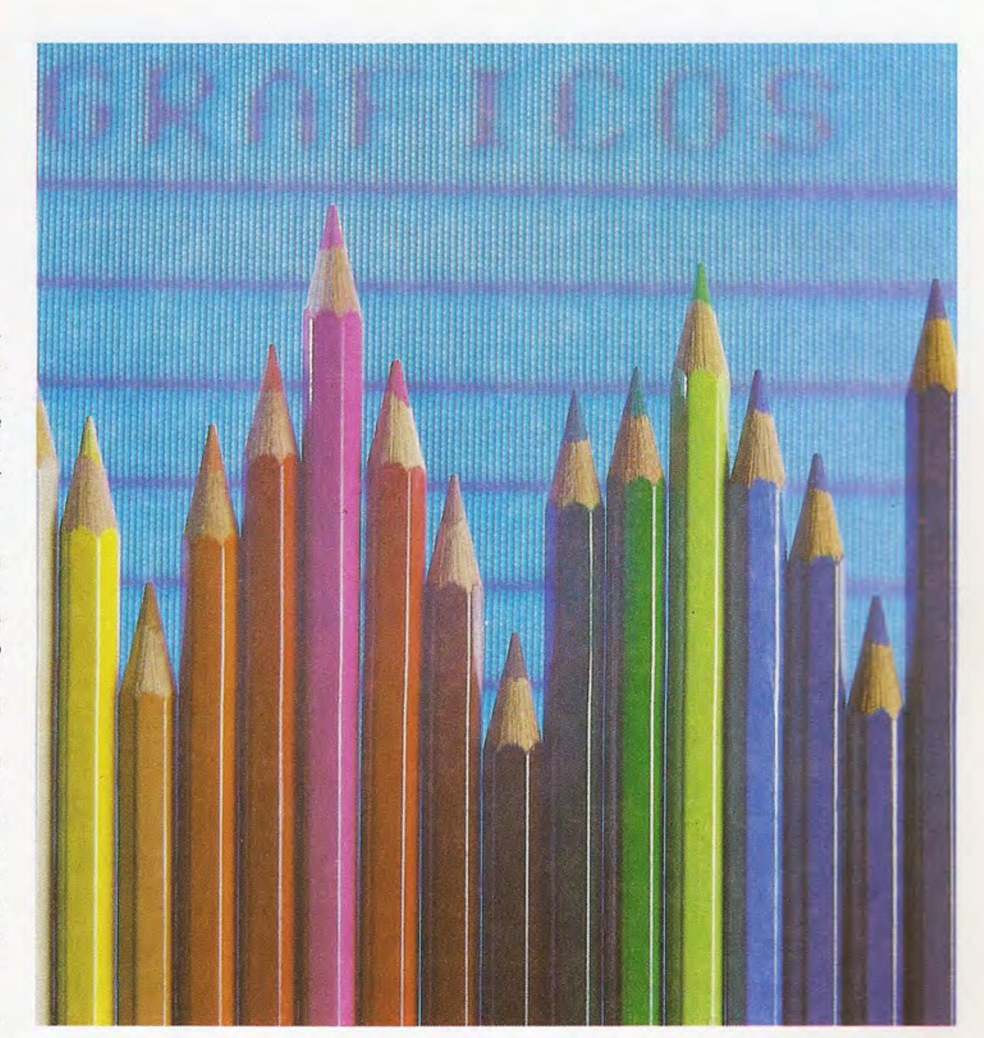
Anteriormente se habló de la imposibilidad de variar la imagen de los caracteres contenidos en ROM. No obstante, puede realizarse un «truco» que permite llevar a cabo la alteración de los caracteres. Para ello es necesario conocer algo más acerca del interior de la máquina.

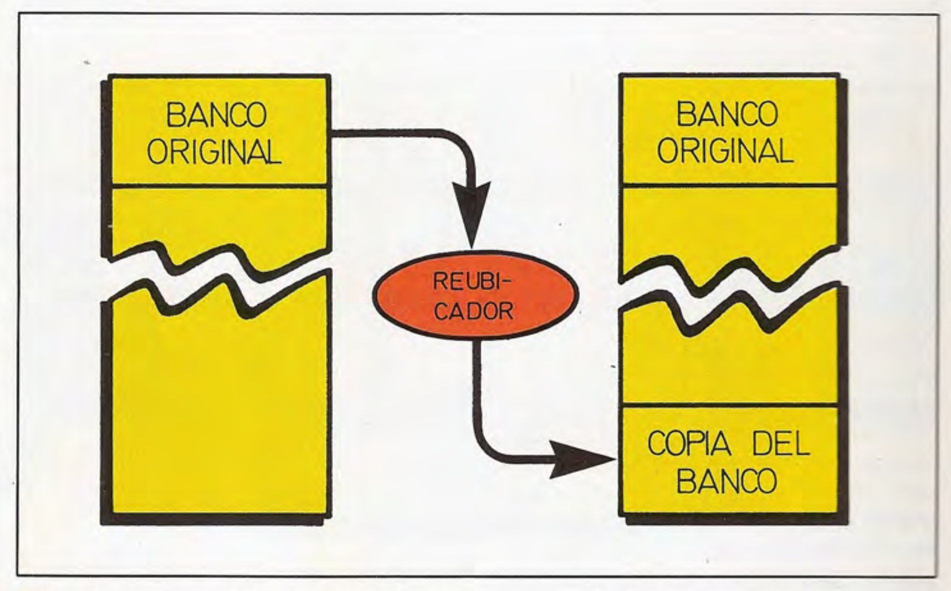
El banco de caracteres se encuentra situado a partir de una determinada posición de memoria. El ordenador accede a la imagen de cada carácter sumando al número que indica esa posición inicial la distancia que separa a la imagen de esa primera posición.

Por ejemplo, sea N la primera posición del banco de caracteres. Para acceder al primer carácter se suma N+0 dando lugar a la posición primera de ese carácter. Para el segundo, se ha de calcular N+8 (ya que las 8 primeras posiciones pertenecen al primero). En general, para determinar la posición del carácter número X se ha de calcular N+(8*X).

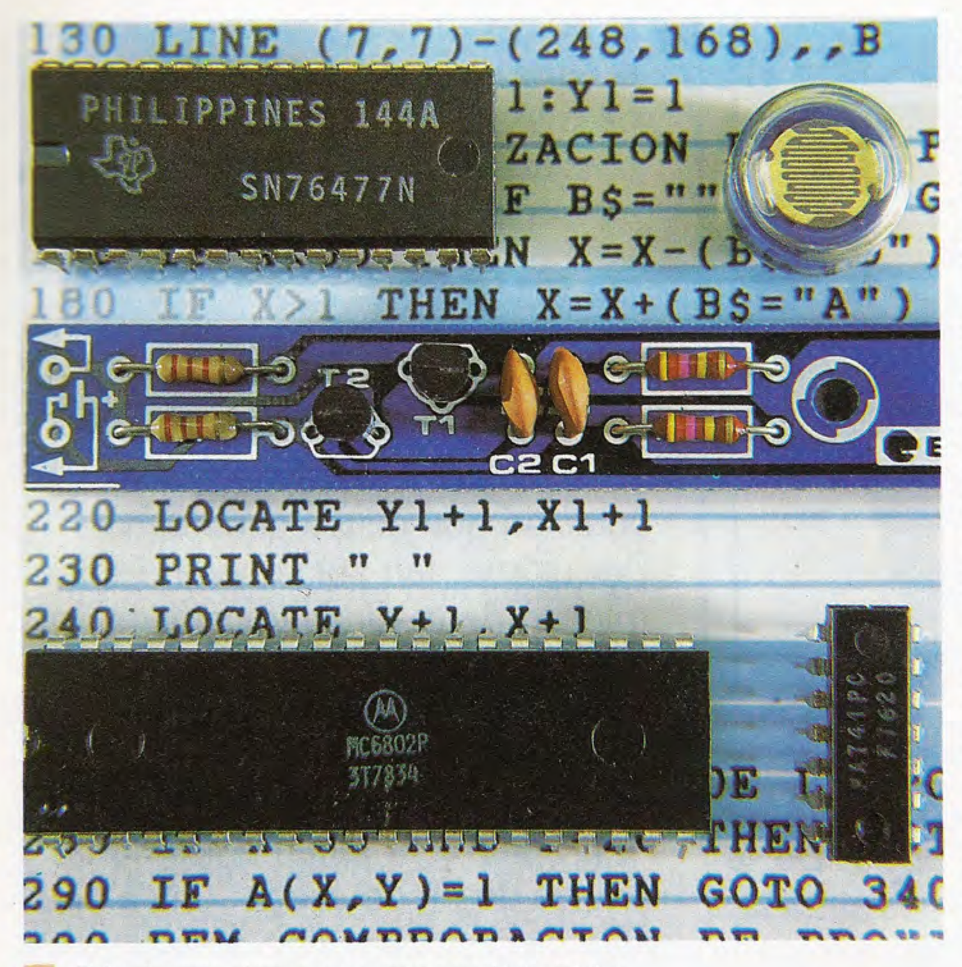
El número que indica el inicio del banco de caracteres (el denominado N) suele estar almacenado en una de las variables del sistema. Estas se encuentran en RAM y, por lo tanto, es posible alterarlas. Una vez identificada la variable del sistema que contiene esa información, ésta se puede modificar por el uso del comando POKE. Si se altera esa variable, el ordenador buscará las imágenes de los caracteres en otro lugar de la memoria, lo que proporcionará unas imágenes completamente aleatorias de los mismos. Ello puede remediarse, claro está; e incluso aprovechar esta posibilidad para nuestro objetivo, cual es modificar el repertorio de caracteres del ordenador.

Para hacer un uso provechoso de la referida variable del sistema, es conveniente que a partir de la dirección introducida en ella exista una copia del banco de caracteres. Los pasos a seguir son los siguientes: identificar una posición de memoria que sea propicia, copiar el





El programa reubicador descrito en el texto permite efectuar una copia del banco de caracteres en otra zona de memoria.



El fundamento técnico que otorga a cualquier equipo doméstico sus amplias y versatiles facultades gráficas, se encuentra en los «chips».

Algunos ordenadores disponen de un juego de caracteres semigráficos que permiten realizar dibujos sencillos.

banco de caracteres a partir de dicha posición y alterar al contenido de la variable del sistema para que apunte a la nueva dirección.

El primer paso revela que no vale cualquier dirección. Ello se debe a la longitud del banco de caracteres. Si el ordenador dispone de un repertorio de 125 caracteres, el banco ocupará 125 * 8 = 1024 bytes (ocho bytes por cada carácter). Para copiar el banco se necesitará, pues, una zona de 1024 bytes libres. Esto significa que la nueva posición de memoria debe estar al comienzo de una zona RAM de 1024 bytes, y que la alteración de dichos bytes no debe producir ningún efecto desastroso (desde luego, dicha zona no debe coincidir con la destinada al almacenamiento del programa, a las variables del sistema, etc.).

Una vez determinada la posición de memoria idónea, habrá que situar a partir de ella las imágenes de los caracteres. Esto último se puede realizar por medio de un programa BASIC. En realidad, lo único que ha de hacer el programa es copiar el contenido de unas posiciones de memoria en otras. A continuación se muestra una posible rutina adecuada para copiar del banco de caracteres.

```
10 REM REUBICADOR DE CARACTERES
20 INPUT "POSICION ORIGINAL";PO
30 INPUT "NUEVA POSICION";NP
40 FOR I=0 TO 1023
50 POKE NP+I,PEEK(PO+I)
60 NEXT I
```

Tras la ejecución de esta rutina se dispondrá de un juego de caracteres alternativo a partir de la posición de memoria introducida como respuesta al comando INPUT de la línea 30. El bucle FOR hace variar a I de 0 a 1023. Con ello, la expresión PO+I recorre los 1024 bytes del banco de caracteres. La función PEEK proporciona el contenido de cada una de esas posiciones de memoria, valor éste que se deposita en la correspondiente nueva posición NP+I mediante el comando POKE de la línea 50.

Inmediatamente después de la ejecución de la rutina reubicadora se puede cambiar el contenido de la variable del sistema por el valor de la nueva posición. Ello permite conmutar el banco antiguo por el nuevo. La reubicación y

posterior conmutación de bancos da como resultado un nuevo banco de caracteres en RAM. O lo que es lo mismo: un banco de caracteres alterables. La programación de dichos caracteres puede ya efectuarse siguiendo las normas comentadas en los anteriores párrafos.

Uso de los caracteres programados

En algunos ordenadores resulta ocioso el método de la reubicación del banco de caracteres. Ello se debe a que en algunos aparatos (como es el caso del Spectrum de Sinclair) una parte de los caracteres se almacena en memoria RAM. Con esto se permite su fácil programación por parte del usuario. El método a seguir para alterar esos caracteres es idéntico al que se utiliza tras el proceso habitual de la reubicación.

Los caracteres programados son fácilmente utilizables, sin más que introducirlos en el argumento de un comando PRINT. Esto permite posicionarlos en cualquier lugar de la pantalla. También es posible su movimiento, por el método de ir imprimiéndolos en posiciones sucesivas de la pantalla.

Por regla general, los caracteres programados se agrupan para formar una figura mayor. En ese caso, cada carácter contendrá sólo una parte de la figura final. A la hora de su representación, las distintas partes se sitúan contiguas en el argumento de PRINT. Si las cuatro cuartas partes del dibujo de una cara se almacenan en los caracteres A, B, C y D, podrían utilizarse las dos líneas PRINT que siguen para visualizar correctamente el resultado:

```
10 PRINT "AB"
20 PRINT "CD"
```

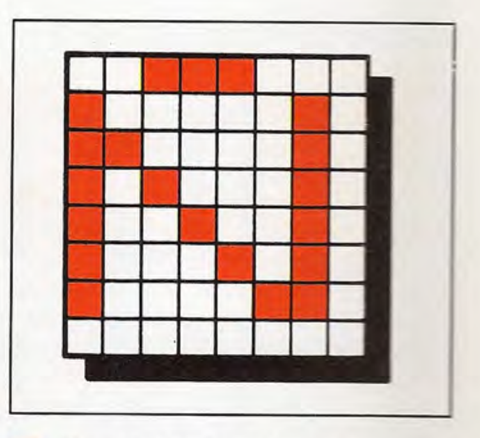
Para situar el dibujo en cualquier punto de la pantalla se hace uso de la opción AT asociada al comando PRINT (o del comando LOCATE en su caso).

Si se tienen los valores que indican la fila y la columna en las variables Y y X, respectivamente, puede emplearse la siguiente rutina posicionadora:

```
100 PRINT AT(X,Y); "AB"
110 PRINT AT(X,Y+1); "CB"
```

1	Ø	Ø	Ø	Ø	Ø	1	Ø
1	1	Ø	Ø	Ø	Ø	1	Ø
1	Ø	1	Ø	Ø	Ø	1	Ø
1	Ø	Ø	1	Ø	Ø	1	Ø
1	Ø	Ø	Ø	1	Ø	1	Ø
1	Ø	Ø	Ø	Ø	1	1	Ø
1	Ø	Ø	Ø	Ø	Ø	1	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

La forma de cada carácter está en correspondencia con un determinado número de unos y ceros almacenados en memoria.



Una posible aplicación de los caracteres definibles es la creación de caracteres especiales no existentes en el repertorio original del ordenador; tal es el caso de la ñ del castellano.

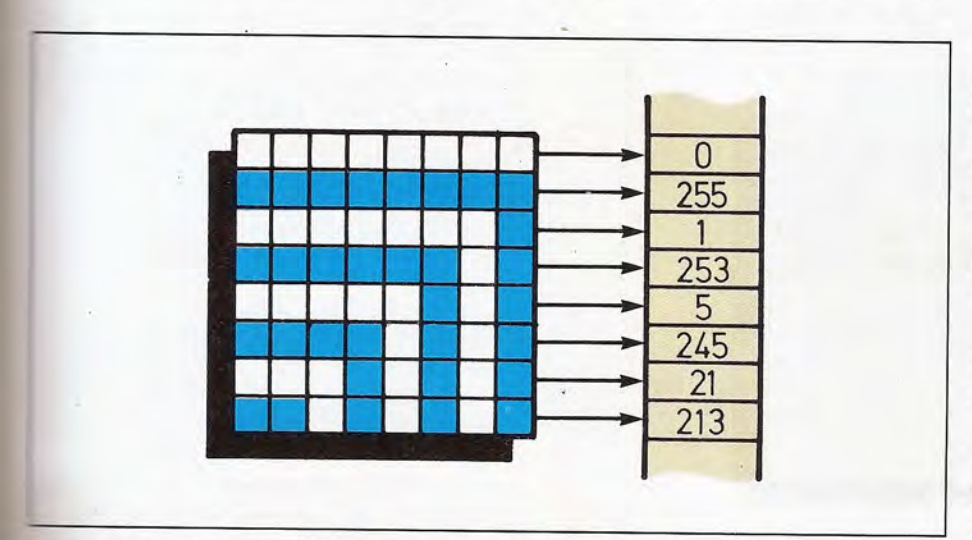
Sprites

La redefinición o programación de caracteres permite crear y utilizar con facilidad figuras móviles. Sin embargo, esta técnica tiene sus limitaciones. Una de ellas es el hecho de que un carácter sólo puede situarse en la intersección de una fila y una columna de texto. Esto hace que el movimiento más suave posible se realice en saltos de ocho pixels (anchura de cada carácter).

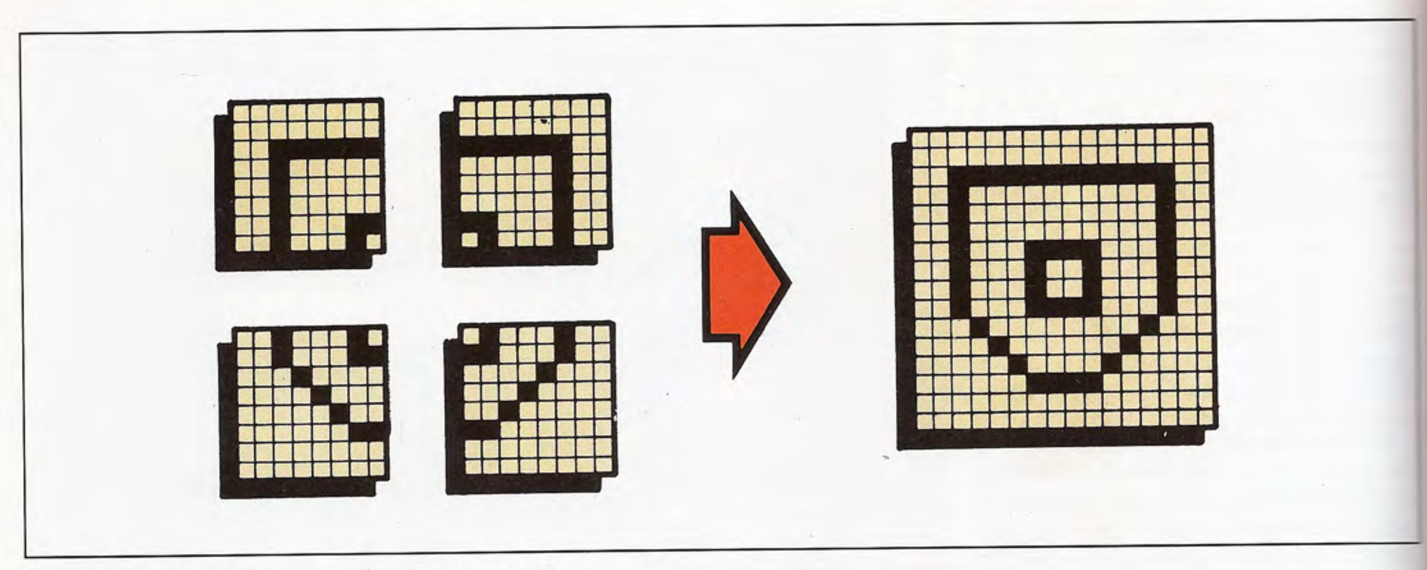
Para solventar este y otros problemas, en los más modernos ordenadores se hace uso de una nueva posibilidad: los denominados «sprites».

Un sprite no es más que un carácter programable mejorado. Estre esas mejoras cabe destacar la posibilidad de movimiento pixel a pixel y la detección de colisiones.

El dialecto BASIC que mejor emplea esta característica es el de Microsoft, el



Cada fila de un carácter ocupa una posición de memoria. En consecuencia, un carácter completo nececita 8 bytes de memoria para su almacenamiento.

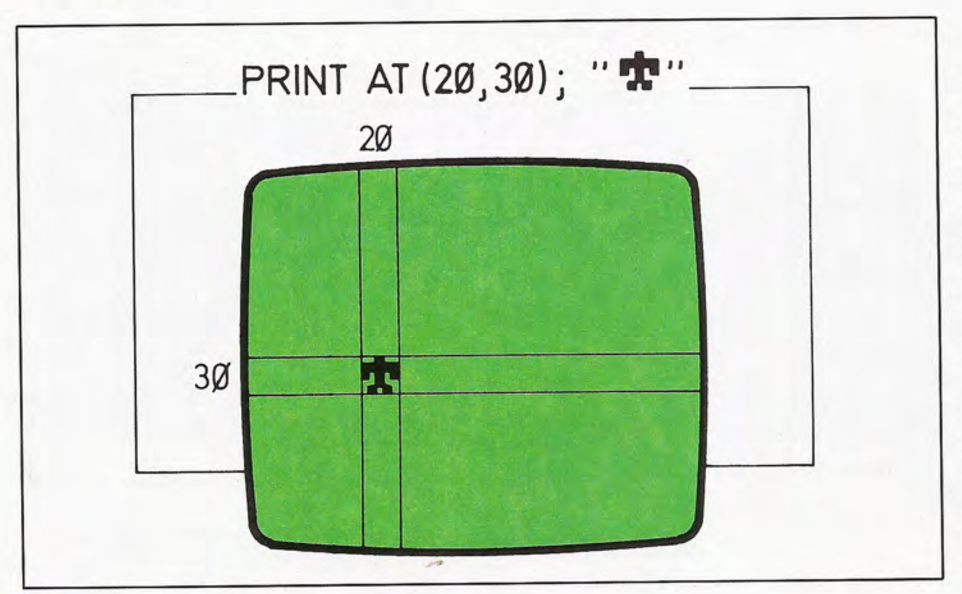


La agrupación de varios caracteres puede utilizarse para construir dibujos de mayores proporciones.

cual equipa a los ordenadores de tipo MSX. Ese será pues el que se comente brevemente en estas páginas.

La definición de la forma de un sprite es idéntica a la definición de un carácter en lo que se refiere a la transición pixels-bits. Sin embargo, en el BASIC de Microsoft esos bytes no se han de almacenar directamente en la posición de memoria, sino en una variable alfanumérica. Dicha variable alfanumérica se corresponde con uno de los elementos del array SPRITE$( ). Cada elemento, desde SPRITE$(0) hasta SPRITE$(255), almacena una imagen. El siguiente es un ejemplo válido de definición de un sprite:

```
  10 FOR I=0 TO 7
  20 READ A$
  30 B$=B$+CHR$(VAL("&B"+A$))
  40 NEXT I
  50 SPRITE$(0)=B$
1000 DATA 00111100
1010 DATA 01111110
1020 DATA 11011011
1030 DATA 11011011
1040 DATA 11111111
1050 DATA 00111100
1060 DATA 01000010
1070 DATA 10000001
```



Para utilizar en la pantalla los caracteres definidos basta con hacer uso del comando PRINT.

Este método tan exótico es el que se utiliza para la creación de sprites. La forma se define a base de un bloque de instrucciones DATA. Estos datos se convierten al formato adecuado por medio de la serie de funciones CHR$(VAL("&B"+_ aplicadas a la ristra de unos y ceros que definen las filas del sprite. La información de las diferentes filas se concatena (+) en la variable B$ y ésta se guarda posteriormente en SPRITE$, creándose así el sprite.

Una vez definido, el sprite puede ser ubicado en cualquier parte de la pantalla mediante el comando PUT SPRITE. Este comando admite en su argumento el número del sprite a representar (en el ejemplo es el sprite número cero) acompañado de las coordenadas (en alta resolución) que señalan su posición. Otro dato aportable es el color del sprite.

Introducción al sonido

Generación de sonidos con el ordenador



La revolución microinformática es un hecho incontestable. En los últimos años estamos asistiendo a una asombrosa proliferación de microordenadores. Una avalancha de equipos cuyo objetivo no es tan sólo el de automatizar aplicaciones que podríamos llamar «profesionales» o «serias». Hoy en día, el ordenador también se está convirtiendo en un electrodoméstico habitual en muchos hogares.

El éxito de los ordenadores domésticos se debe a varios factores. En primer lugar, al convencimiento de que el futuro está en los ordenadores y es necesario familiarizarse con ellos para no quedar descolgado. En segundo lugar están las aplicaciones informáticas que podríamos llamar «menores»: llevar pequeñas contabilidades, procesar textos, aplicaciones educativas y muchas otras. Por último, y con una importancia capital se encuentran los juegos. En gran medida, estos pequeños ordenadores han sustituido a las consolas de videojuegos y a otros divertimentos como instrumentos de ocio y diversión para mayores y pequeños.

Definiendo términos

Al igual que sucede con otras características inherentes a los equipos informáticos, también en lo relativo al sonido existe una gran confusión, sobre todo por lo que a terminología se refiere. En los próximos párrafos se describirán algunos de los conceptos claves cuyo conocimiento resulta imprescindible para el programador.

¿Qué es el sonido? La respuesta no es difícil: se trata de perturbaciones que se propagan a través del aire y que nuestro oído es capaz de detectar. Estas perturbaciones se pueden crear de muy diversas formas: por medios mecánicos, por nuestra voz... y también por medios electrónicos. Para lograr tal objetivo, los especialistas emplearon unos circuitos electrónicos llamados osciladores, capaces de generar ondas de una determinada frecuencia; ondas que se harán audibles al reproducirlas a través de un altavoz.

Cuando estas ondas se encuentran dentro de la gama de frecuencias audible por el ser humano, se convierte en lo que llamanos sonido.

La frecuencia, como se puede apreciar, es uno de los parámetros más importantes de una onda. Se trata de una medida de la velocidad a la que se producen las variaciones de esa perturbación. Sin embargo, estas ondas no sólo se caracterizan por su frecuencia, sino también por otros factores como son la forma de la onda, su amplitud y la potencia de la misma.

¿Qué significa que un ordenador posea tres canales de sonido y uno adicional de ruido? Definamos en primer lugar qué se entiende por canal.

Un canal de sonido se refiere a una fuente sonora; de tal forma que un ordenador con tres canales de sonido poseerá tres fuentes emisoras de sonido. Ello no quiere decir que posea tres altavoces, sino que el sonido se genera mediante tres osciladores, cuyas salidas, luego, son mezcladas y enviadas, por lo general, a un solo altavoz. El canal de ruido se concreta en otro oscilador que genera una serie de ondas, aunque esta vez de una forma un tanto peculiar. Este último resulta adecuado para crear sonidos efectistas: lo que podríamos denominar efectos especiales.

Dentro del dispositivo que produce el sonido es preciso distinguir dos partes fundamentales: los osciladores, que producen el sonido, y los amplificadores y altavoces que lo acondicionan para que seamos capaces de oírlo.

Hay ordenadores que en su interior incluyen ambas cosas; pero también los hay que prescinden de la segunda zona. En este último caso, el sonido se obtiene aprovechando los amplificadores y altavoces del televisor o monitor conectado al equipo. Al efecto, la señal que el ordenador envía al televisor no es sólo la de vídeo, sino también la de sonido.

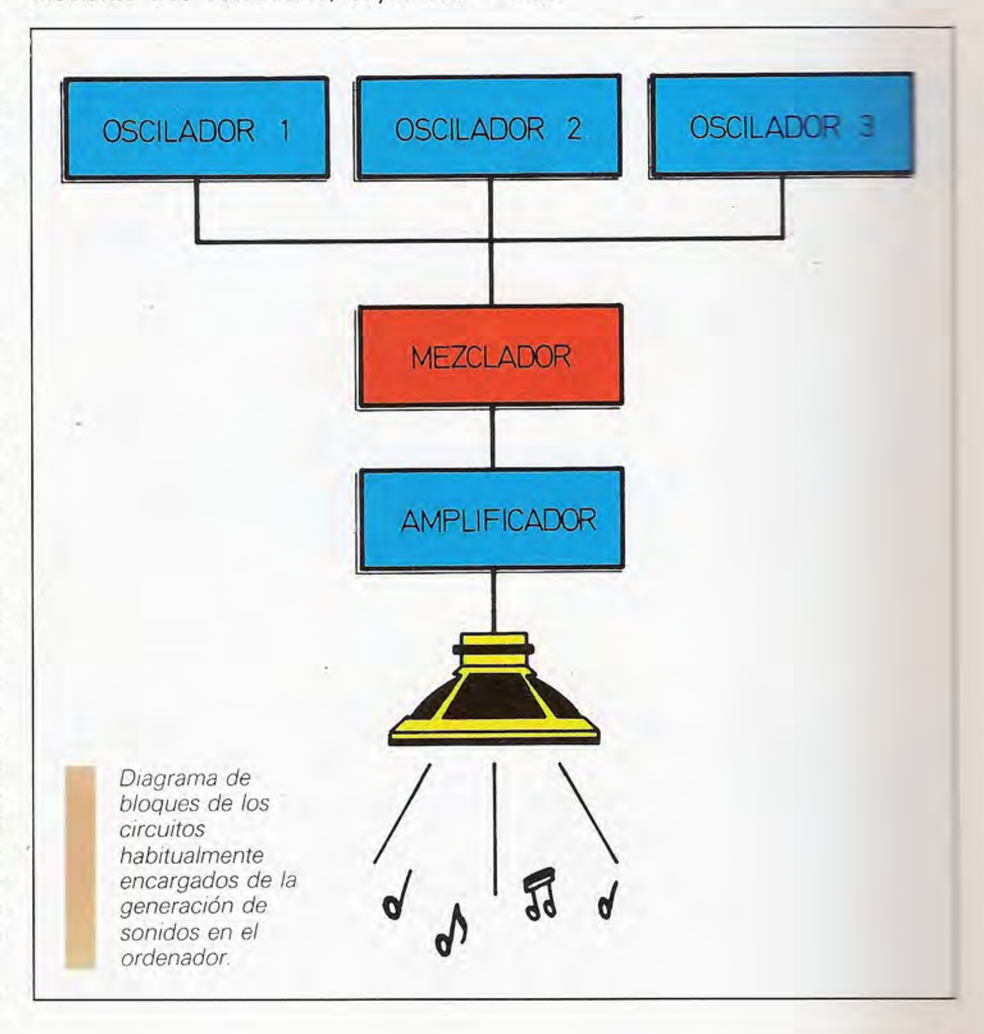
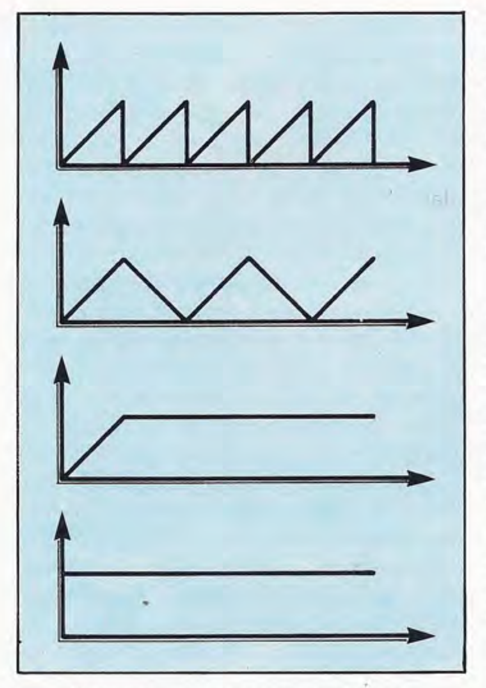


Diagrama de bloques de los circuitos habitualmente encargados de la generación de sonidos en el ordenador.

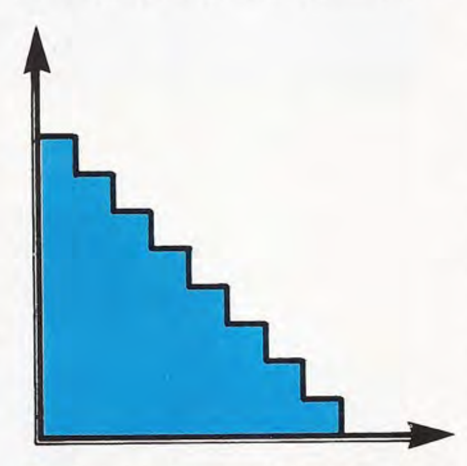


Distintas formas o patrones de la envolvente de volumen que pueden utilizarse para la síntesis de sonidos especiales.

Respecto a la reproducción final del sonido cabe puntualizar algunos detalles de interés.

Para empezar, los altavoces que suelen incorporar los ordenadores son generalmente pequeños y emiten un sonido de poca importancia. Además, este sonido no suele ser regulable; esto es: no se puede ajustar el volumen del mismo una vez que éste ha sido programado. En los ordenadores que aprovechen el amplificador y el altavoz del televisor o monitor externo, la cosa cambia; éstos poseen, por lo general, un mando para controlar el volumen, mando que el usuario puede regular a voluntad.
La diferencia fundamental, de todas formas, reside en la calidad. En el caso de que sonido se obtenga a través del receptor de TV, tiene lugar toda una serie de procesos de codificación y decodificación de la información sonora que hacen que el sonido resultante pierda calidad. También existen ordenadores que disponen de salidas para equipos de alta fidelidad (a veces en estereofonía) de los cuales cabe esperar un sonido excelente.

Son muchos los términos del lenguaje musical que se emplean en el campo de los ordenadores. Por ejemplo, una nota es un sonido de una determinada frecuencia. A su vez, una octava es un conjunto de frecuencias. Cuando se dice que un ordenador tiene más octavas que otro, se está indicando que el rango de frecuencias en el que puede trabajar es más amplio. Las notas musicales que caben dentro de una octava son esencialmente siete; aunque, de hecho, estas notas admiten variantes de tal forma que dentro de una octava hay en realidad más de siete sonidos. Por lo general, para emitir una determinada nota hay que indicar a la máquina de qué nota se trata y en qué octava se encuentra. El nombre de la nota será el mismo en cualquiera de las octavas, depende tan sólo de su posición relativa dentro de la misma.



En un ordenador, el nivel de volumen del sonido es regulable en sucesivos «escalones» o saltos de forma digital.

Hay que tener en cuenta otro detalle de importancia a la hora de leer el manual que acompaña a cada ordenador. Habitualmente, se utiliza una escala musical en la que cada nota recibe un nombre: DO, RE, MI, FA, SOL, LA y SI. No obstante, la notación que suele encontrarse en los manuales (normalmente en inglés) no es ésta. Se trata de la misma escala, pero con la diferencia de que los nombres de las notas aparecen como: C, D, E, F, G, A y B. Esta es la notación habitual anglosajona.

Esbozando sonidos

Las instrucciones para el manejo del sonido, al igual que ocurre con las destinadas al tratamiento de gráficos, no están en absoluto estandarizadas. Este hecho hace que los diferentes ordenadores presenten instrucciones muy diversas para el manejo del sonido. Incluso ciertos ordenadores carecen de instrucciones BASIC de esta categoría, lo que obliga a recurrir al acceso directo o memoria.

Dentro del microordenador, suele existir algún circuito integrado al que se le encomienda la misión de apoyar las tareas de generación sonora. A este «chip» tan sólo hay que completarlo con algún dispositivo que amplifique la pequeña señal que proporciona, antes de mandar la señal resultante al altavoz.

Antes se indicó que tal amplificación no suele realizarse en el propio ordenador, sino que la señal se envía al televisor para que éste se encargue de amplificarla.

Para comunicar al circuito integrado «experto en sonido» la información que éste precisa para realizar su tarea, se utilizan unas determinadas posiciones de memoria. Posiciones a las que tiene acceso el generador de sonido y de donde toma los datos que debe depositar en sus registros.

Empezaremos hablando del caso en el que es necesario acceder directamente a la memoria principal del ordenador para programar los circuitos que éste posee para producir sonidos. Tal operación es realizable en cualquier aparato.

BEEP

Genera un sonido con una duración y una frecuencia indicadas en su argumento.

Formato: BEEP <duración>,<frecuencia>

Ejemplos:
```
10 BEEP 0.5,7
50 BEEP 1,5
```



En ciertos microordenadores, la impresión del carácter ASCII 7 no produce alteración alguna en la pantalla, sino tan sólo la emisión de un pitido.

No obstante, siempre que sea posible se huirá de ello; por la sencilla razón de que el procedimiento resulta un tanto complicado. En principio, suele ser necesario acceder a una posición de memoria para fijar el nivel de volumen deseado. Acto seguido, es preciso acceder a otras posiciones para fijar la nota, la duración, la envolvente, etc. El problema reside en que para ello es necesario recordar las posiciones de memoria asociadas a cada uno de estos registros, y los valores adecuados que han de introducirse en los mismos. Además, en muchos casos estos valores están relacionados con la frecuencia de emisión por medio de ecuaciones matemáticas más o menos complicadas.

Una alternativa sencilla para obtener sonidos consiste en acudir a una serie de palabras BASIC, que al ser interpretadas por el ordenador dan lugar a un sonido determinado y fijo, no admitiendo por lo tanto ningún parámetro. Tal es el caso, por ejemplo, de la palabra BEEP. Esta da nombre a un comando cuya ejecución hará que el altavoz emita un determinado sonido durante un cierto intervalo de tiempo. También existen otras palabras como ZAP, EXPLODE, etc. Dentro de esta misma categoría de herramientas para generar sonidos elementales cabe considerar el carácter ASCII 7; un carácter de control que produce un pitido audible al introducirlo en el ordenador a través de una instrucción PRINT:

```
PRINT CHR$(7)
```

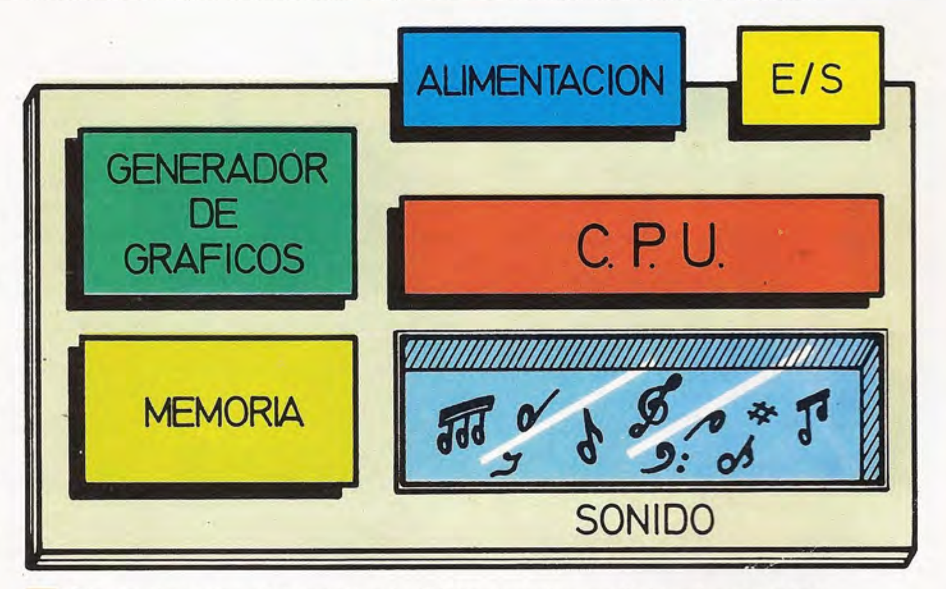
Avanzando en el sonido

El lenguaje BASIC da entrada a otro tipo de intrucciones más elementales para la programación de sonidos. Como ya suele ser habitual —aunque lamentablemente—, las coincidencias en el repertorio de los distintos intérpretes BASIC en este punto son casi inexistentes.

Vamos a empezar hablando de la instrucción BEEP, pero esta vez acompañada por dos parámetros. Uno de ellos servirá para indicar la frecuencia de la nota deseada y el otro señalará la duración que se desea que tenga el sonido. En este caso no será necesario seleccionar la octava deseada, sino que bastará con dar números cada vez mayores con lo que se irá pasando de una octava a la siguiente.

Uno de los problemas inherentes a este tipo de instrucciones es que al emplear números se pierde de vista la nota y octava con la que se está trabajando. No obstante, el principal problema reside en que cuando se ejecuta esta instrucción el programa se detiene hasta que se concluye su tratamiento. Así, pues, el sonido no es independiente del proceso.

Este sistema es el que utiliza un ordenador tan popular como el ZX-Spectrum. A continuación se incluyen, a título de ejemplo, algunos programas destinados a este ordenador.



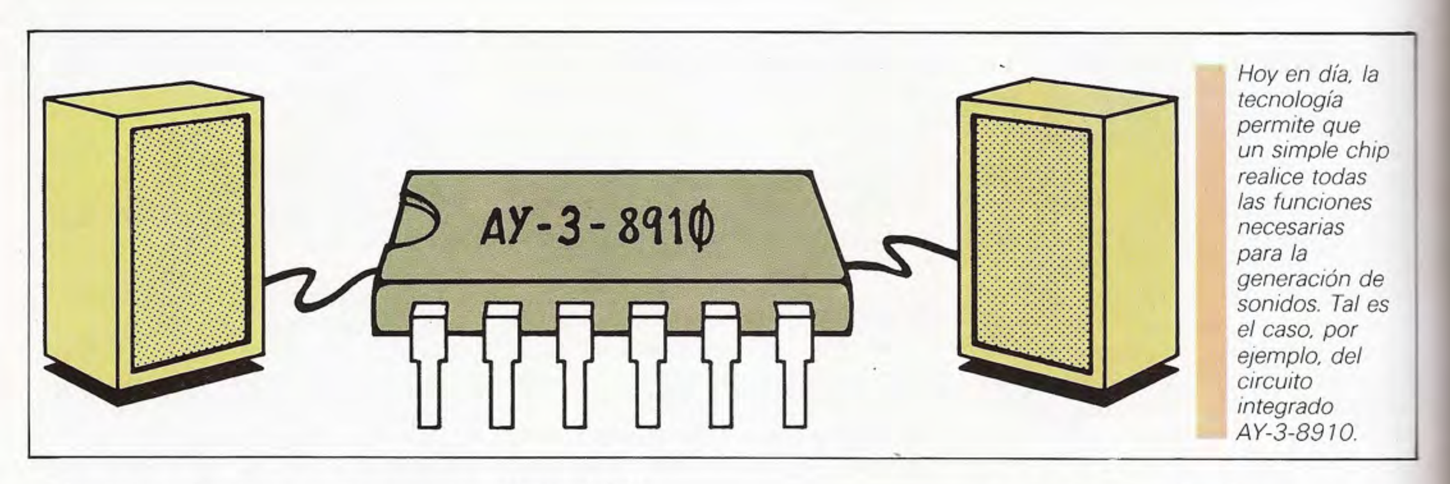
Dentro de la compleja interioridad del ordenador, también caben circuitos especializados en la generación de sonidos.

SOUND

Permite acceder a los registros de sonido para su programación.

Formato: SOUND <registro>,<contenido>

Ejemplos:
```
20 SOUND 5,3
70 SOUND 11,128
```



Hoy en día, la tecnología permite que un simple chip realice todas las funciones necesarias para la generación de sonidos. Tal es el caso, por ejemplo, del circuito integrado AY-3-8910.

El primero de ellos permite recorrer una octava completa:

```
10 FOR I=0 TO 11
20 BEEP 0.5,I
30 NEXT I
```

El mismo programa puede ejecutarse para una octava más alta, lo que permitirá observar la diferencia entre las notas reproducidas en una u otra octava:

```
10 FOR I=0 TO 11
20 BEEP 0.5,I+12
30 NEXT I
```

Análogamente, la octava seleccionada puede ser una inferior:

```
10 FOR I=0 TO 11
20 BEEP 0.5,I-12
30 NEXT I
```

Cuando se trata de definir una composición relativamente larga, es conveniente, en lugar de escribir tantas instrucciones BEEP como notas se deseen interpretar, encerrar esta instrucción dentro de un bucle. La lectura de los datos se realizará mediante instrucciones READ/DATA; en esta últimas estarán escritos los valores que conducirán al resultado apetecido. Veamos un ejemplo:
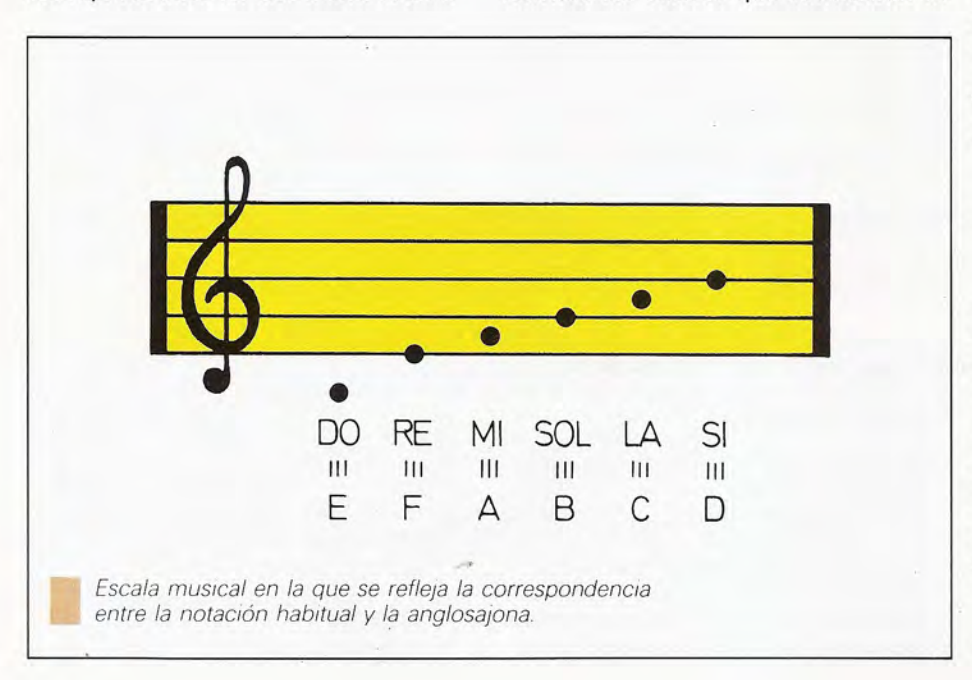
```
100 FOR I=1 TO 100
110 READ A,C
120 BEEP A/2.50,C
130 NEXT I
140 DATA 0.25,7,0.25,9,0.5,11,0.5,14,0.75,14,0.25,
    16,0.5,14,0.5,11,0.75,7
```

Al terminar de leer este bloque de instrucciones el ordenador se detendrá presentando un mensaje de error, debido a que no hay suficientes datos para solventar todas las pasadas que hay que dar al bucle. Esta breve rutina se ha confeccionado de forma que sea utilizable para reproducir composiciones de cualquier longitud, sin más que añadir sucesivas líneas DATA con los datos oportunos. Cuando la composición esté terminada, será suficiente con contar los datos que la integren y ajustar el valor final del bucle en función de dicho número.

Acceso a los registros de sonido

Al principio del capítulo se mencionó que ciertos ordenadores exigían el acceso directo a los registros de sonido para programar la producción del mismo. Por otra parte, también existen ordenadores con instrucciones BASIC que facilitan el trabajo directo con los registros de sonido, sin necesidad de recurrir a los laboriosos POKES. Veamos un ejemplo práctico, utilizando el chip de sonido AY-3-8910; uno de los más empleados en los modernos ordenádores.

Si el ordenador posee más de un canal de sonido, las opciones se multipli-



Escala musical en la que se refleja la correspondencia entre la notación habitual y la anglosajona.

TABLA DE CONVERSION

Ordenador	Sonidos fijos		Sonidos variables	Acceso a los registros
	Comandos directos	CHR$ (7)	BEEP n1, n2	SOUND <r>, <c>
AMSTRAD	—	CHR$ (7)	SOUND n1, n2, n3, n4, n5, n6, n7(4)	—
APPLE II (APPLESOFT)	—	CHR$ (7) o CTRL+G	—	—
APRICOT (M-BASIC)	—	—	—	—
ATARI	—	—	SOUND n1, n2, n3, n4 (1)	—
CBM 64	—	—	—	—
DRAGON	—	—	SOUND n2, n1	—
EQUIPOS MSX	BEEP	CHR$ (7)	—	SOUND <r>, <c>
HP-150	—	CHR$ (7)	—	—
IBM PC	—	CHR$ (7)	SOUND n2, n1	—
MPF	—	—	—	—
NCR DM-V (MS-BASIC)	—	CHR$ (7)	—	—
NEW BRAIN	—	—	—	—
ORIC	SHOOT, ZAP, PING, EXPLODE	CHR$ (7) o CTRL+G	MUSIC n1, n2, n3 (2)	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	—
SINCLAIR QL	—	—	BEEP (3)	—
SPECTRAVIDEO	BEEP	CHR$ (7)	—	SOUND <r>, <c>
ZX-SPECTRUM	—	—	BEEP n1, n2	—

n1: duración. n2: nota. <r>: registro. <c>: contenido.

(1) SOUND n1, n2, n3, n4. n1: canal. n2: nota. n3: distorsión. n4: volumen.

(2) MUSIC n1, n2, n3, n4. n1: canal. n2: octava. n3: nota. n4: volumen.

(3) BEEP [n1, n2[, n3, n4, n5[, n6[, n7[, n8]]]]]. n1: duración. n2: nota. n3: nota secundaria; entre ella y n2 oscilará el sonido. n4: intervalo entre osciladores de notas. n5: define el tamaño de cada oscilación. n6: repetición. n7: rizado. n8: aleatorio.

(4) SOUND n1, n2, n3, n4, n5, n6, n7: n1: canal. n2: período. n3: duración. n4: volumen. n5: envolvente de volumen. n6: envolvente de tono. n7: período de ruido.

can. Existen instrucciones previas que seleccionan uno y otro canal; e incluso instrucciones que admiten más parámetros, uno para cada canal. Veamos un ejemplo de esto último:

```
BEEP (2,4), (3,1), (2,4)
```

en donde cada uno de los paréntesis se refiere a un canal.

El referido chip dispone de 14 registros accesibles por el programador. Cada uno de estos registros contiene información específica que es utilizada por el circuito integrado para saber el sonido que ha de generar. La tabla adjunta incluye una lista de los referidos registros, señalando las funciones asociadas a cada uno de ellos.

Frecuencia

Los registros del 0 al 5 se utilizan por

pares para determinar la frecuencia del sonido a emitir. Se rigen por la siguiente fórmula:

```
X=(1.78977*10 EXP 6)/16*F
RB=INT (X/256)
RA=X—RB*256
```

en donde RA puede coincidir con uno de los registros 0, 2 ó 4, y RB con los registros 1,3 ó 5.

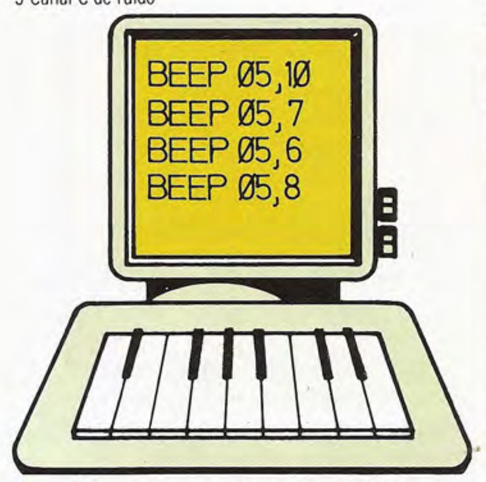
Ruido

El registro seis selecciona la frecuencia central del ruido, por medio de un valor que ha de ser buscado en la correspondiente tabla; en ella, cada frecuencia tiene asignado un valor específico.

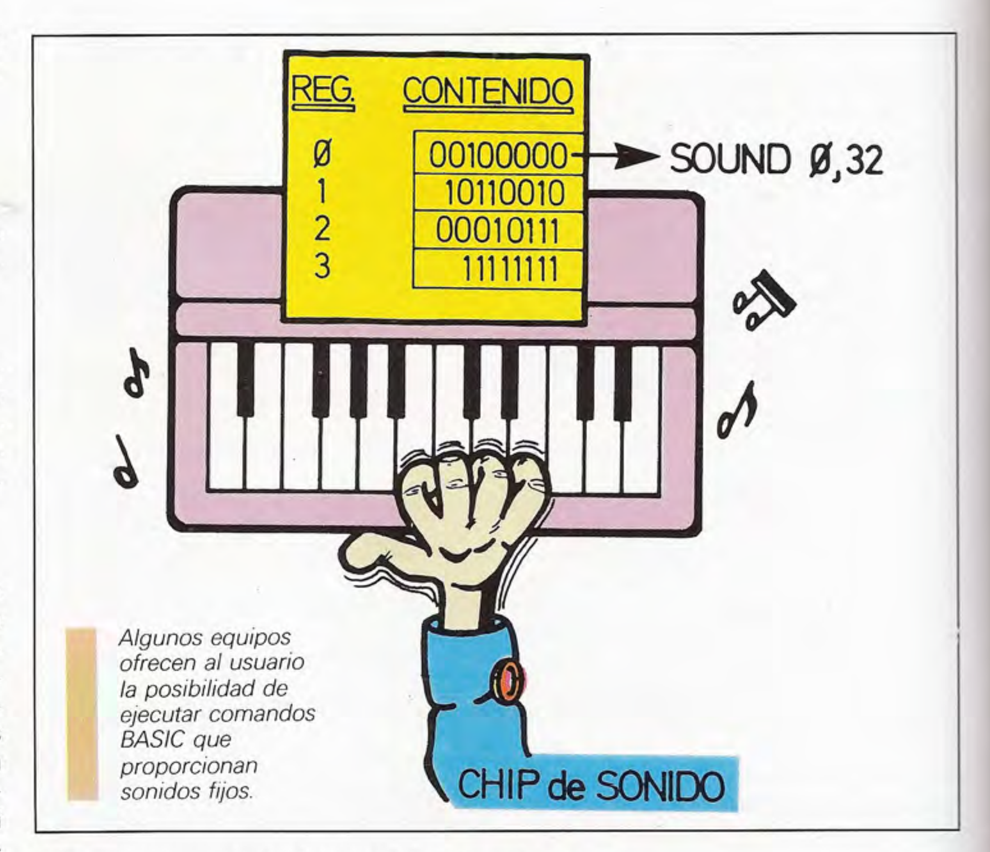
Selección del tipo de canal

El registro al efecto se emplea para indicar al generador el uso que se va a dar a cada uno de los canales. Cada canal puede ser utilizado para emitir sonido o bien ruido. Para efectuar esta selección se utilizan los seis bits menos significativos del registro. Según esté a uno o a cero cada uno de estos bits, indicará si el canal correpondiente está seleccionado o no. La siguiente tabla revela la utilidad de los seis bits del registro. La tabla comienza por el bit menos significativo y termina con el sexto bit (el último que se utiliza).

0 Canal A de sonido
1 Canal B de sonido
2 Canal C de sonido
3 Canal A de ruido
4 Canal B de ruido
5 Canal C de ruido



El empleo del comando BEEP con parámetros permite conseguir muy diversos tipos de sonidos.



Algunos equipos ofrecen al usuario la posibilidad de ejecutar comandos BASIC que proporcionan sonidos fijos.

Evidentemente, el siguiente paso consiste en transformar la palabra binaria obtenida a notación decimal, e incluirla en la instrucción oportuna.

Volumen

Los registros del 8 al 10 preseleccionan el volumen con el que ha de emitir cada uno de los canales de sonido. Sus valores van del 0 al 15. Cuando su valor es 16, el control pasa al generador de envolventes.

Envolvente

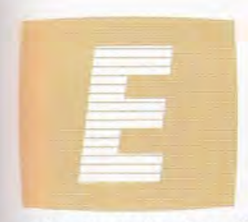
La envolvente define la forma en la que cambia el nivel de volumen del sonido producido. El chip que nos ocupa tiene ocho patrones diferentes, cuyo aspecto puede observarse en la figura que acompaña al texto.

Los registros 1 y 12 fijan el período de la envolvente, mientras que el registro 3 sirve para seleccionar el patrón de la misma.

REGISTROS DEL CHIP DE SONIDO AY-3-8910			
Registro	**Función**		
0	Frecuencia canal A	Control preciso	0-255
1	Frecuencia canal A	Control de escala	0-18
2	Frecuencia canal B	Control preciso	0-255
3	Frecuencia canal B	Control de escala	0-18
4	Frecuencia canal C	Control preciso	0-255
5	Frecuencia canal C	Control de escala	0-18
6	Frecuencia de ruido		0-31
7	Programación de tono de ruido para cada canal		0-63
8	Volumen del canal A		0-16
9	Volumen del canal B		0-16
10	Volumen del canal C		0-16
11	Frecuencia de la envolvente	Control preciso	0-255
12	Frecuencia de la envolvente	Control de escala	0-255
13	Envolvente de ataque		0-255

Los sonidos del BASIC

El macrolenguaje musical

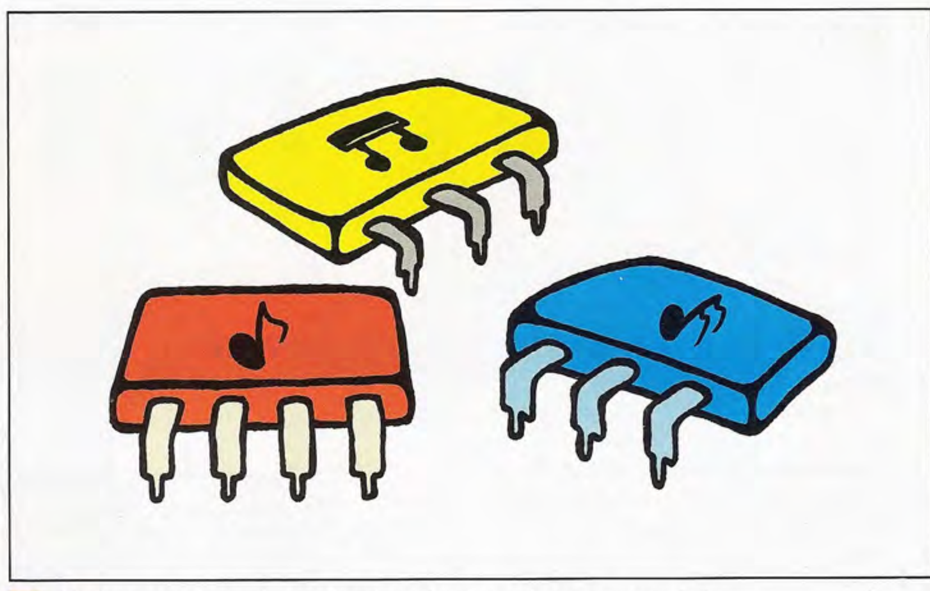


El control del sonido, al igual que el de los gráficos, no posee un tratamiento estandarizado en todos los dialectos del BASIC. La razón hay que achacarla a los distintos criterios que adoptan los fabricantes a la hora de elegir el circuito generador de sonido. Dicho circuito suele admitir un número limitado de órdenes elementales que son, precisamente, las indicadas en el primer capítulo dedicado al sonido.

El circuito sonoro del ordenador puede ser programado para producir una gran variedad de tonos y melodías. Algunos aparatos incorporan un pequeño repertorio de sonidos preprogramados. Estos sonidos son accesibles mediante sencillos comandos BASIC, con lo que el empleo de la «voz del ordenador» se simplifica en gran medida.

El presente capítulo está dedicado a una peculiar forma de utilizar sonidos preprogramados. Se trata del denominado «macrolenguaje musical», presente en la mayor parte de los ordenadores que emplean el BASIC de Microsoft. El macrolenguaje musical es una potente herramienta para la creación de las más variadas melodías.

Su estructura y forma de uso es similar a la del comentado macrolenguaje gráfico.



El manejo del sonido en el ordenador depende en gran medida del circuito especializado que incluya el equipo.

Macrolenguaje musical y comando play

El macrolenguaje musical va asociado al comando PLAY, el cual permite el acceso a dicho macrolenguaje. El comando PLAY es el encargado de hacer que se ejecuten las órdenes del macrolenguaje incluidas en su argumento; órdenes que se expresan dentro de una cadena de caracteres.

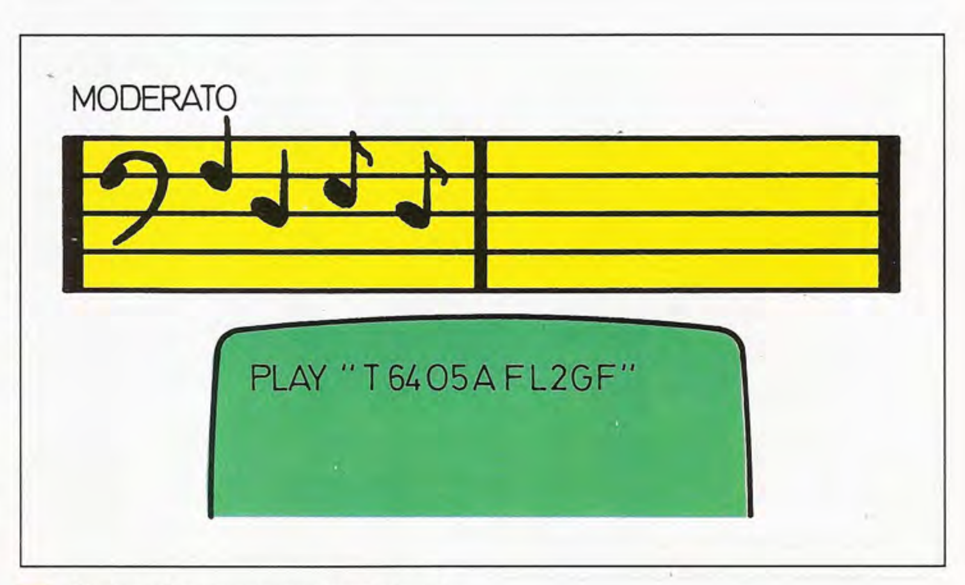
La ejecución de una instrucción PLAY hace que se envíen las órdenes pertinentes al circuito de sonido del ordenador.

Una vez recibidas, el generador de sonido se pone a trabajar independientemente del resto del ordenador. Esto significa que mientras se genera el sonido se puede seguir ejecutando el programa principal. Con ello se evita que la ejecución del programa se vea retardada por efecto de la emisión de sonido.

Como ya se ha comentado, el comando PLAY admite una cadena de caracteres en su argumento. Dicha cadena debe ajustarse a las normas impuestas por el macrolenguaje. En definitiva, el formato que ha de mostrar el comando PLAY es el siguiente:

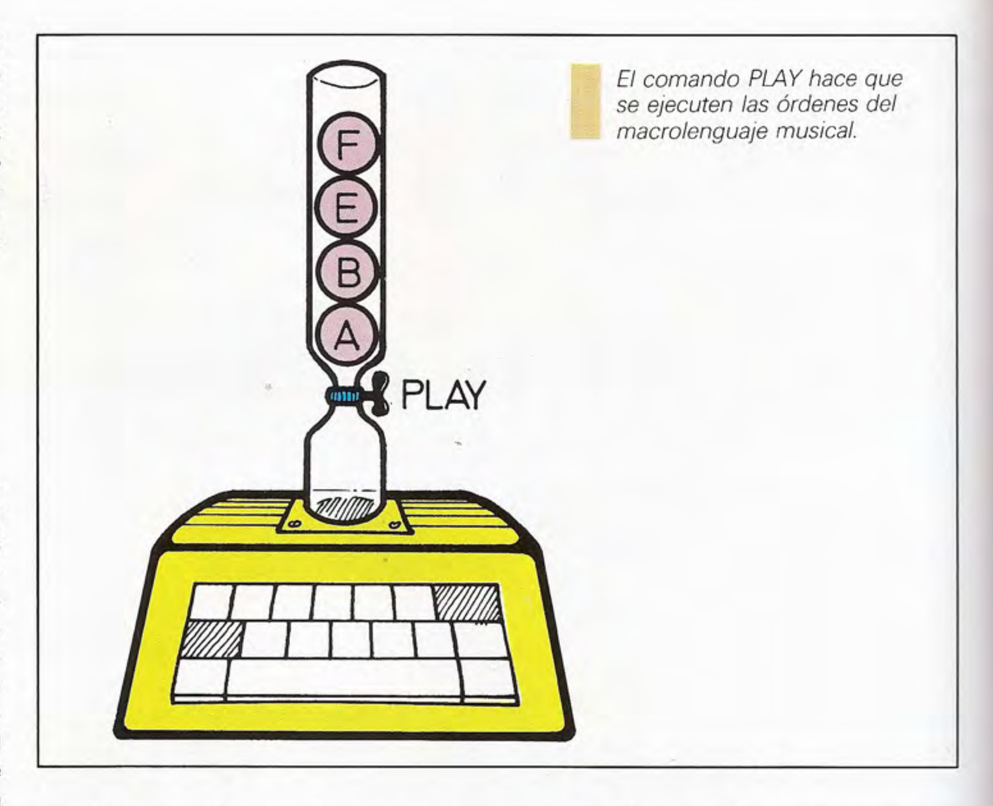
(Número de línea) PLAY <cadena1> [, <cadena2>...]

En el formato general de PLAY se aprecia la posibilidad de añadir más de



El macrolenguaje musical adopta una nomenclatura muy cercana al cifrado clásico de la música.

una cadena. Ello es posible cuando el aparato con el que se está trabajando dispone de más de un canal de sonido; esto es, permite emitir más de una voz al tiempo. En algunos ordenadores es corriente el empleo de hasta tres canales de voz, lo que hará posible la creación de una melodía con hasta dos acompañamientos.



El comando PLAY hace que se ejecuten las órdenes del macrolenguaje musical.

Primeros pasos: las notas

El macrolenguaje musical de Microsoft adopta una filosofía que será del agrado de aquellos que posean algunos conocimientos de solfeo. En primer lugar, los sonidos se indican por sus correspondientes notas. De todos son conocidas las siete notas de la escala musical: DO, RE, MI, FA, SOL, LA y SI. Sin embargo, esa no es la nomenclatura utilizada aquí. El macrolenguaje musical emplea la signatura anglosajona, que tiene una correspondencia biunívoca con la anterior. Esta nomenclatura hace uso de las siete primeras letras del abecedario inglés (las mismas del castellano sin la «ch»). La relación entre ambas formulaciones se muestra en la correspondiente tabla.

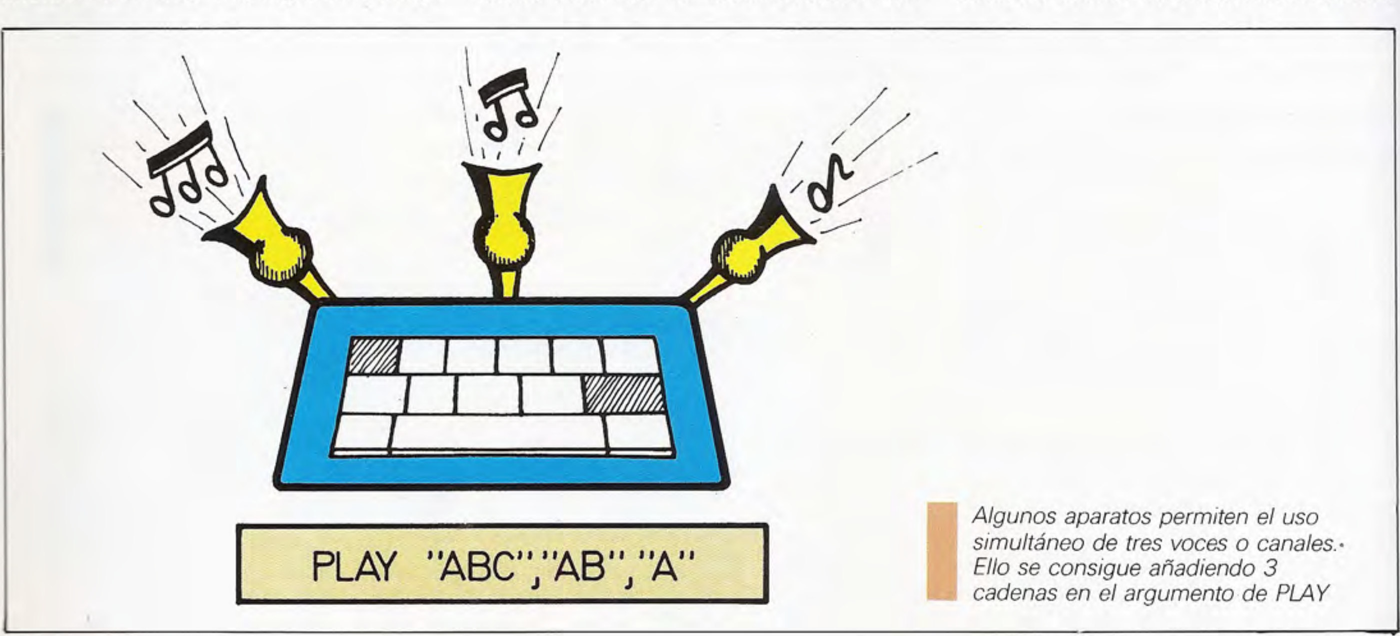
Como se puede observar en la tabla adjunta, la nueva nomenclatura comienza en la nota LA, siguiendo luego en el orden habitual. Así pues, para la interpretación de una escala de DO a SI basta con ejecutar la línea que se indica a continuación.

```
10 PLAY "CDEFGAB"
```

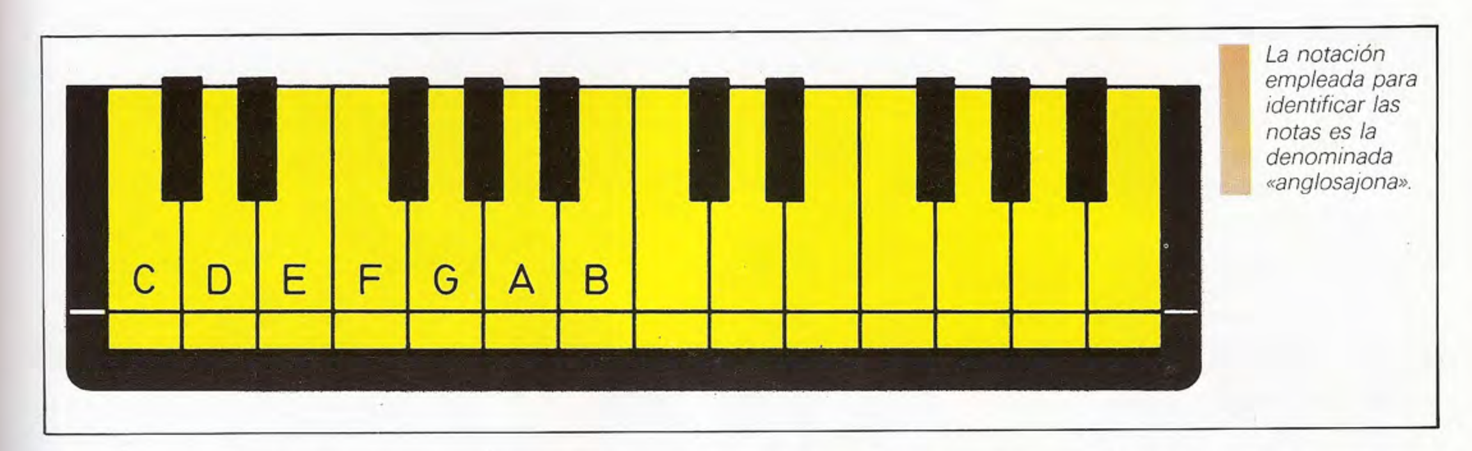
En el caso de disponer de varios canales, cada uno de ellos admitirá una cadena diferente. En ese supuesto, las distintas cadenas irán separadas por comas. La misma escala entonada a tres voces se formularía como sigue:

```
10 PLAY "CDEFGAB", "CDEFGAB", "CDEFGAB"
```

Al interpretar las tres voces la misma melodía, ésta se oirá como si se tratara



Algunos aparatos permiten el uso simultáneo de tres voces o canales. Ello se consigue añadiendo 3 cadenas en el argumento de PLAY



La notación empleada para identificar las notas es la denominada «anglosajona».

de una sola voz. Para apreciar con mayor claridad cada una de las voces o canales conviene indicar distintas melodías para cada una. Ejecutando el siguiente ejemplo se emitirán notas diferentes por cada canal:

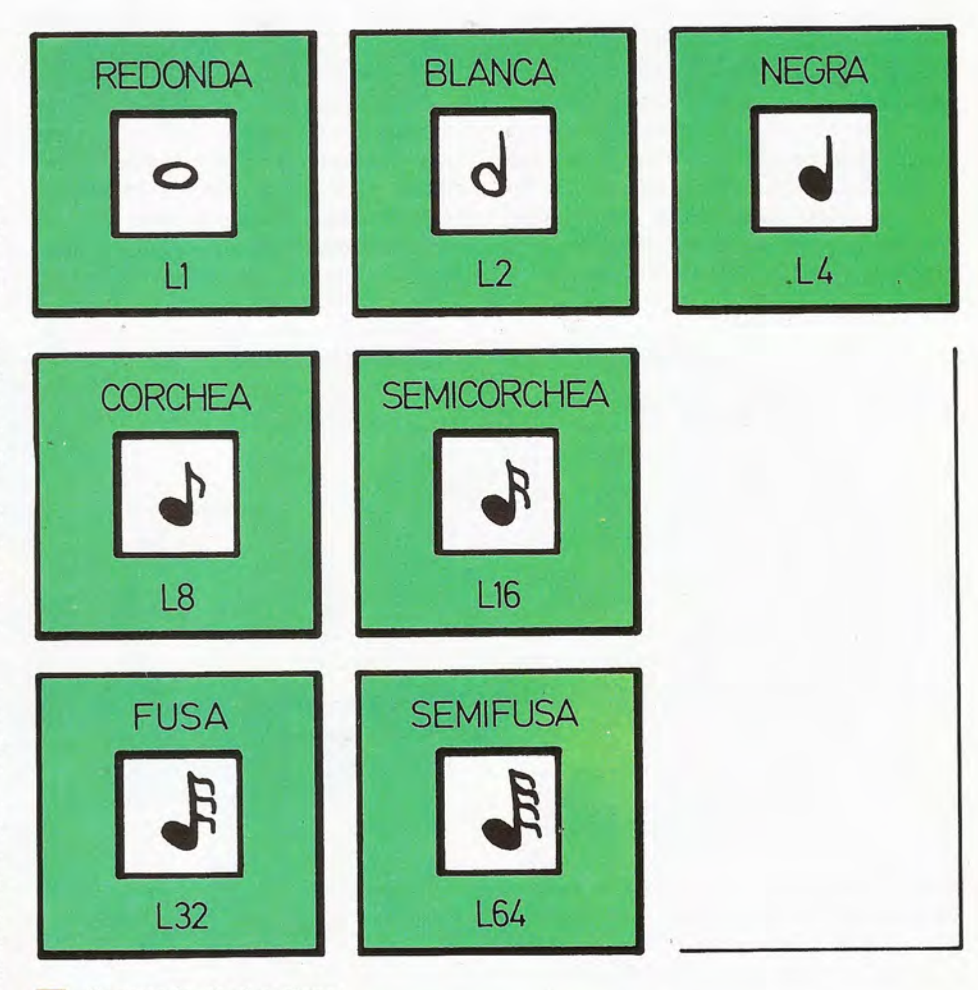
```
10 PLAY "CDEFGAB", "DEFGABC", "EFGABCD"
```

Sostenidos y bemoles

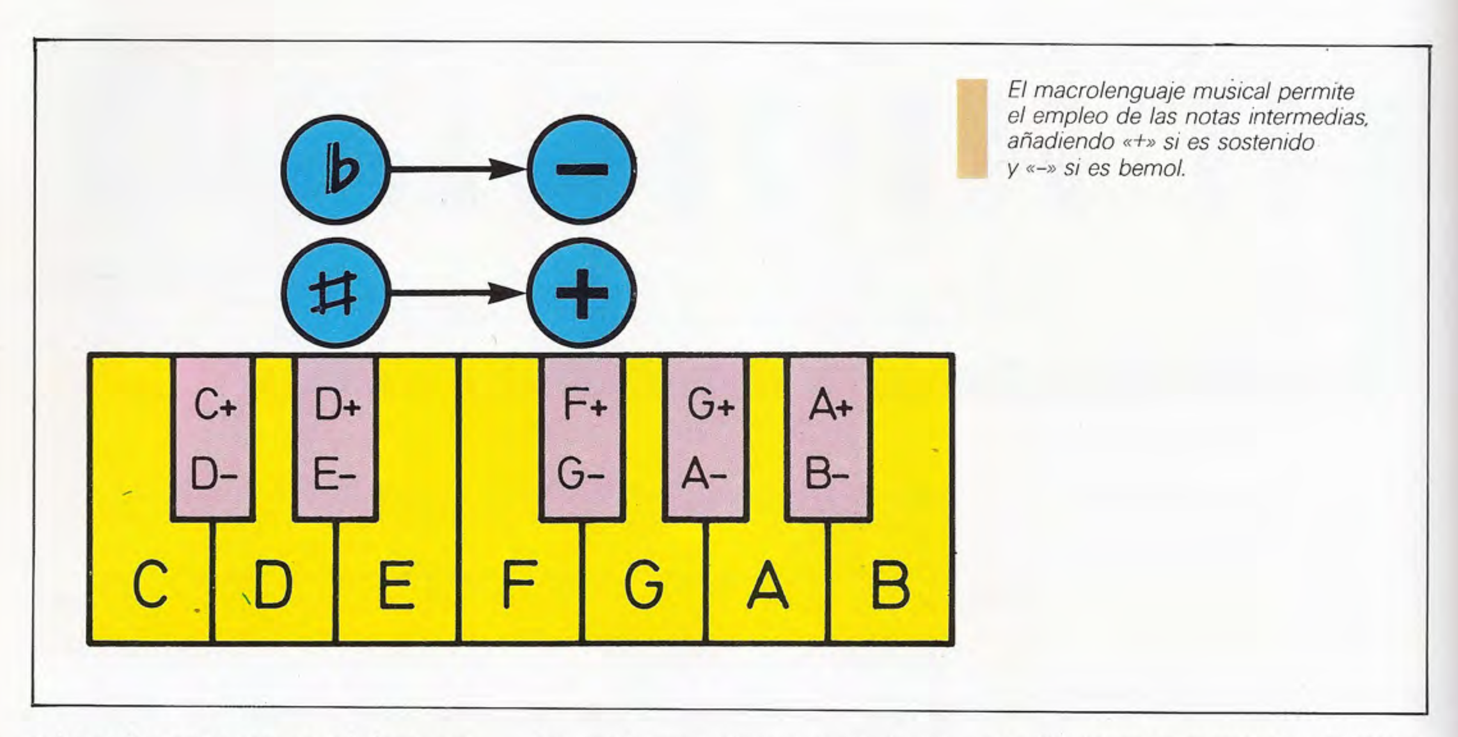
Las siete notas de la escala cromática no son las únicas existentes. En realidad, la separación entre las sucesivas notas sigue una regla un tanto extraña. La diferencia entre dos notas recibe el nombre de tono o semitono dependiendo de las notas que sean. Un tono indica una separación en frecuencia fija, de la cual el semitono es justamente la mitad. Para aclarar conceptos será necesario consultar la tabla de separación entre notas.

Dicho de otra forma, si se separan las notas tomando como unidad el semitono, quedarán huecos libres. Téngase en cuenta que un tono equivale a dos semitonos. La tabla adjunta muestra la escala de semitonos completa.

En ella se observa que quedan posiciones sin ocupar después de cada nota, excepto entre MI/FA y SI/DO. Estos puntos, marcados en la tabla con guiones, corresponden a notas intermedias. Dichas notas se nombran en relación a la nota anterior o posterior. Así, el primer guión de la tabla corresponde al DO sostenido, el segundo al RE sostenido, etc. Esta nomenclatura hace referencia a la nota inmediatamente anterior. Si se desea referenciar las notas intermedias en relación a las que la siguen, se utiliza el término «bemol». De esta forma, el primer guión se puede denominar DO sostenido o RE bemol, mientras que el segundo corresponde tanto al RE sostenido como al MI bemol. Las referidas notas intermedias también pueden ser interpretadas mediante el macrolenguaje musical. Para ello se emplean los signos + y – detrás de la nota en cuestión. Por ejemplo, el mencionado primer guión se formularía como C+ (DO sostenido) o D– (RE bemol). Por supuesto,



Repertorio de duraciones estándar de las notas.



El macrolenguaje musical permite el empleo de las notas intermedias, añadiendo «+» si es sostenido y «–» si es bemol.

existen dos notas que no admiten sostenido: MI y SI, ya que sus sostenidos corresponden a FA y DO respectivamente. De la misma forma, ni FA ni DO admiten la correspondiente bemol.

Empleando estas notas intermedias se puede recorrer la escala completa por medio de la siguiente instrucción:

```
10 PLAY "CC+DD+EFF+GG+AA+B"
```

Más escalas: las octavas

Hasta ahora se ha visto la forma de utilizar las notas de una escala. Se sabe que, tras la última nota de una escala, SI, viene el DO de la escala siguiente. En realidad, la secuencia de notas se puede repetir infinitamente, tanto hacia arriba (sonidos agudos) como hacia abajo (sonidos graves). El único límite lo impone la capacidad auditiva del oído humano.

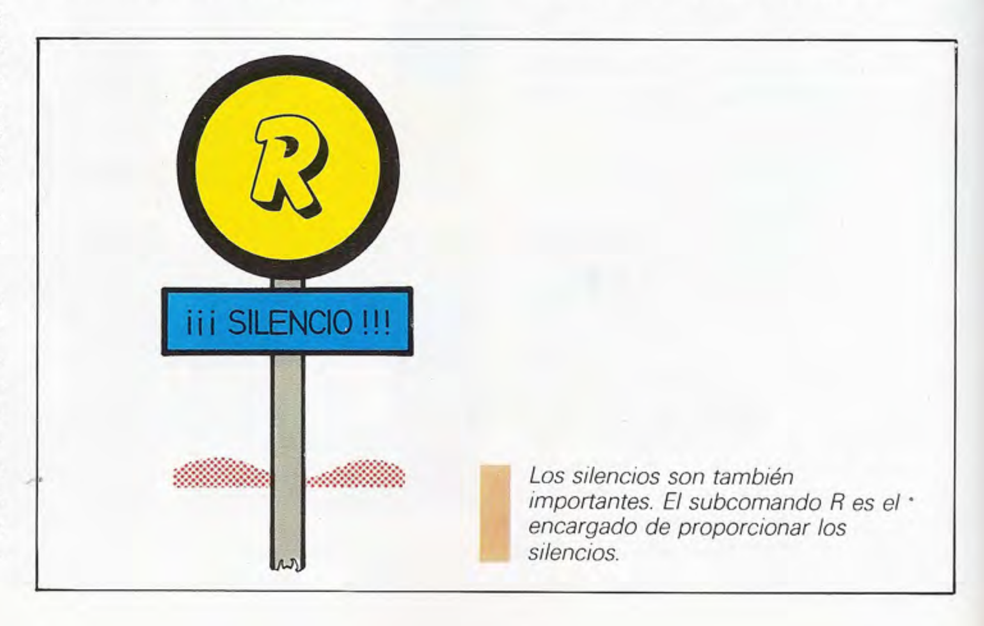
El oído medio es capaz de percibir una gama de frecuencias que supera con creces la extensión de la escala básica. Por ello se añaden más escalas, identificándose cada una por la posición en la que se encuentra. Cada escala individual se denomina «octava». Con esto, un sonido determinado se reconoce por la octava en la que se encuentra, y dentro de la octava, por la nota a la que corresponde.

Los modernos ordenadores incluyen circuitos integrados generadores de sonido capaces de abarcar ocho o más octavas. El macrolenguaje musical contempla esta posibilidad, permitiendo elegir la octava. Para ello se emplea el comando-letra O seguido del número identificativo de la octava deseada. Cuando no se indica la octava (como en los ejemplos anteriores) se utiliza la octava central del espectro. Como ejemplo se muestra el efecto asociado al uso de diferentes octavas para cada canal:

```
10 PLAY "O1CDEFGAB", "O4CDEFGAB", "O7CDEFGAB"
```

Esta sencilla línea hará que suene la misma melodía en tres octavas distintas. Cada canal emplea una octava diferente, sonando las tres al tiempo. Con ello se aprecia tanto la diferencia de octava, como la diferencia de canal. Si se desea que suene cada octava separada (empleando una sola voz) se ha de introducir y ejecutar la siguiente línea:

```
10 PLAY "O1CDEFGABO4CDEFGABO7CDEFGAB"
```



Los silencios son también importantes. El subcomando R es el encargado de proporcionar los silencios.

Tras la ejecución de un comando O, la octava queda fijada hasta que se vuelva a variar por medio de otro comando O. Esto quiere decir que si se ejecuta otro comando PLAY a continuación del incluido como último ejemplo, se comenzará en la séptima octava (07).

Existe otra forma de acceder a las diferentes notas. Este nuevo método no utiliza las escalas habituales, sino que numera las notas consecutivamente. Contando las notas naturales y las intermedias, cada octava tiene doce notas. Si se dispone de un total de ocho octavas, el número de notas permitidas es de 12×8=96.

Las 96 notas pueden identificarse mediante su número de orden. Ese es precisamente el otro modo de acceder a cada nota. Para el empleo de esta notación se hace imprescindible un nuevo comando-letra. Se trata en este caso de la letra N, que seguida por un número del 0 al 95 permite la emisión de la nota correspondiente. En esta notación, N0 corresponde al DO de la primera octava, y N95 al SI de la octava número ocho.

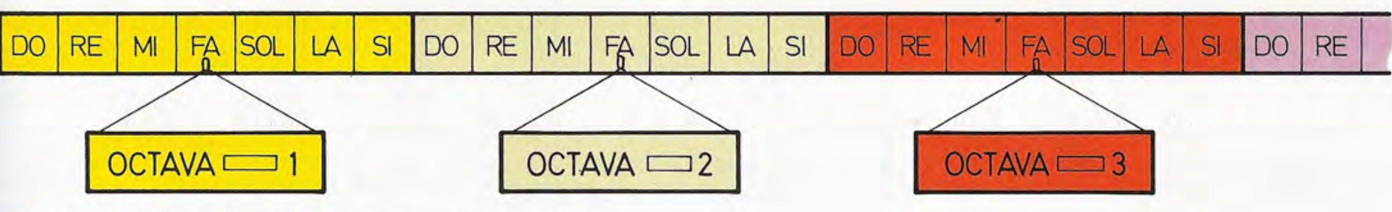
Las dos instrucciones siguientes proporcionan el mismo resultado. La primera emplea la notación habitual, mientras que la segunda hace uso de la notación absoluta:

```
10 PLAY "01CC+DD+EFF+GG+AA+B"
10 PLAY "N0N1N2N3N4N5N6N7N8N9N10N11N12"
```

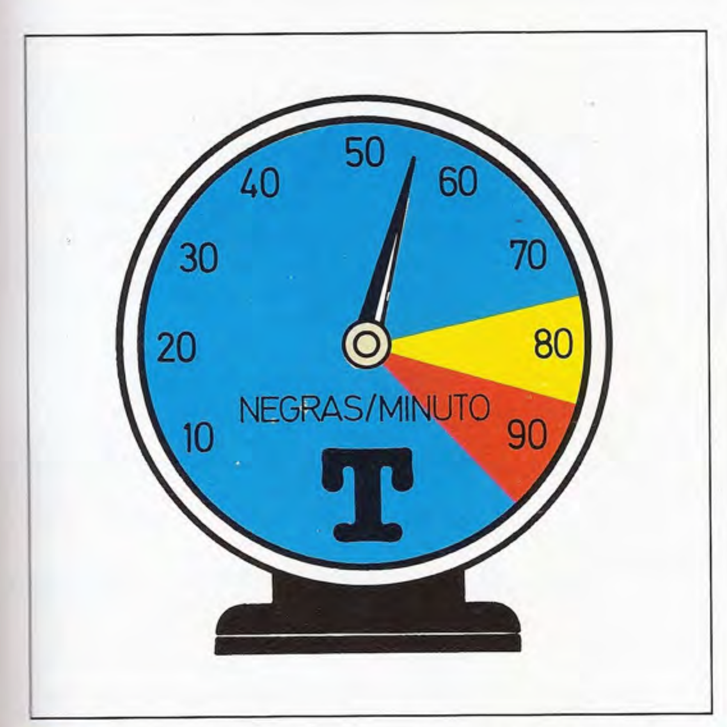
Siguiendo la notación absoluta, el DO de la octava inicial (cuarta octava: O4) se formula como N36.

Duración de las notas

En la interpretación de una melodía la duración de todas las notas no es la misma. Una nota puede mantenerse sonando más o menos tiempo que las demás. Por regla general, a cada nota de una melodía le corresponderá una duración propia. En terminología musical, las duraciones se especifican como submúltiplos de la duración máxima. Cada duración inferior es exactamente la mi-



La repetición cíclica de las escalas obliga a identificar la octava a la que corresponde una determinada nota.



El subcomando T permite variar la velocidad de ejecución de la pieza, prefijando el número de notas negras por minuto que hay que ejecutar.



El volumen del sonido emitido se controla por medio del subcomando V.

tad de la anterior. La nomenclatura empleada para las duraciones de las notas es la que se muestra en la tabla adjunta.

La duración empleada hasta ahora es la de "negra". Esta es la duración que el ordenador toma incialmente. Para hacer que la duración de una determinada nota aumente puede recurrirse a la repetición de la misma. Por ejemplo, la siguiente línea interpreta DO en negra, RE en blanca y MI en redonda.

```
30 PLAY "CDDEEEE"
```

Sin embargo, este método no permite duraciones más cortas. Además, la repetición de una nota no proporciona el efecto exacto: se percibe la separación entre cada dos notas.

En el macrolenguaje musical está prevista la variación de la duración de las notas. Para ello se emplea la letra-comando L.

A continuación de la misma se añade un número que puede variar de 1 a 32. Dicho número indica la fracción de «redonda» que ha de emplearse. La correspondiente tabla relaciona las figuras clásicas con la notación del macrolenguaje.

El comando L permite también especificar duraciones no estándar, como por ejemplo L5 ó L6. A continuación, se muestra un ejemplo en el que la escala se interpreta en corcheas:

Correspondencia entre las nomenclaturas de la escala cromática

DO	C
RE	D
MI	E
FA	F
SOL	G
LA	A
SI	B
DO	C
RE	D
MI	E
FA	F
SOL	G

```
20 PLAY "L8CDEFGAB"
```

En el ejemplo se ve que la acción del comando L queda fijada. Al igual que el comando que especifica la octava (O) su efecto se extiende hasta el punto en el que se ejecuta otro comando análogo. Así, la línea siguiente:

```
30 PLAY "CDEFGAB"
```

tendrá una duración distinta dependiendo de la instrucción, de entre las siguientes, que la preceda:

```
20 PLAY "L1"
20 PLAY "L4"
20 PLAY "L16"
```

Para hacer que cada nota tenga su propia duración puede optarse por preceder cada una de ellas con el correspondiente comando L. En el siguiente ejemplo se interpreta la escala en corcheas, salvo el FA, que tiene duración de semicorchea:

```
20 PLAY "L8CDEL16FL8GAB"
```

Las tres primeras notas están efectadas por el primer comando L8, siendo por lo tanto corcheas. La nota FA se hace semicorchea por medio de L16. Para que el resto de notas siga con la duración inicial, es preciso restaurar ésta con un nuevo L8. En este caso, el proceso es demasiado engorroso para variar la duración de una sola nota. Afortunadamente, el macrolenguaje permite realizar esto mismo de una forma más sencilla.

Si lo que se pretende es alterar la duración de una sola nota, no es preciso el uso del comando L. Como se ha visto, el comando L fija la duración de todas las notas que le siguen. Para especificar la duración de una sola nota basta con indicar esa duración a continuación de la propia nota (se emplea al efecto el número que se utilizaría con el comando L). Haciendo uso de esta nueva facilidad, el anterior ejemplo se reduciría a los siguiente:

```
20 PLAY "L8CDEF16GAB"
```

Se observa en el ejemplo que la variación en la duración sólo afecta a FA. Las restantes notas seguirán con la duración marcada por el último comando L.

La duración de una nota puede ser variada de una tercera forma. En música se emplean puntos para alargar la duración de una nota. Un punto situado inmediatamente después de la nota, alarga ésta en la mitad de su duración. Un segundo punto la alargará en la mitad de la mitad, etc. Esto mismo puede realizarse en el ordenador. El método es

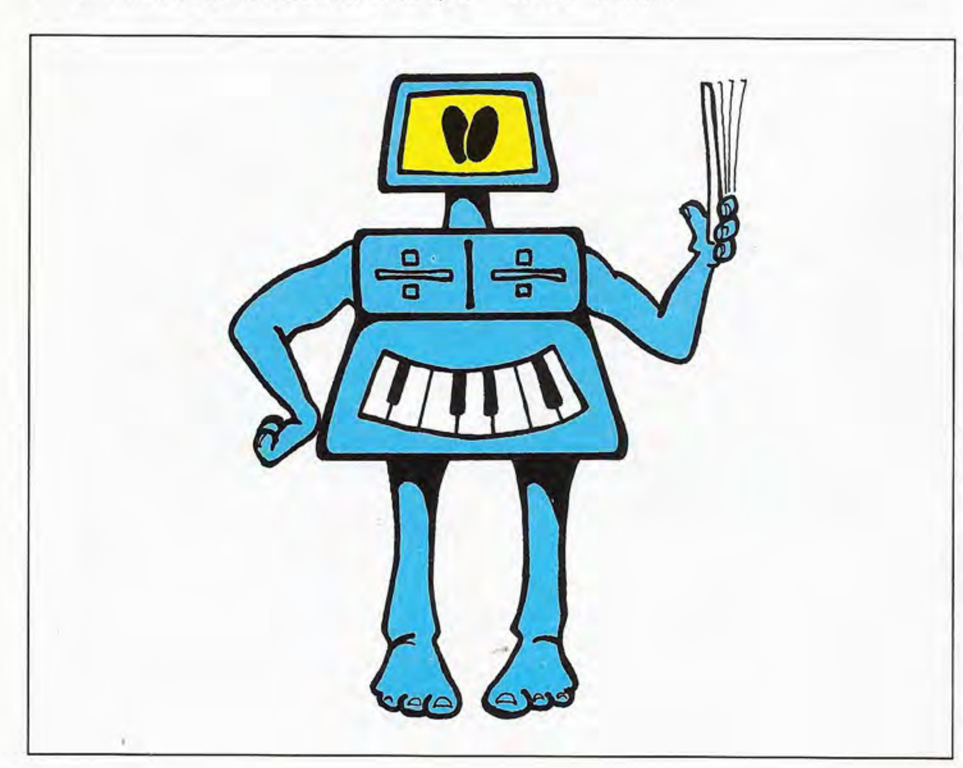


TABLA DE CONVERSION

Ordenador	PLAY	Subcomandos								
	PLAY	A, B, C, D, E, F, G	O	L	R	N	V	T	X <var>;	<com>=<var>;
AMSTRAD	—	—	—	—	—	—	—	—	—	—
APPLE II (APPLESOFT)	—	—	—	—	—	—	—	—	—	—
APRICOT (M-BASIC)	—	—	—	—	—	—	—	—	—	—
ATARI	—	—	—	—	—	—	—	—	—	—
CBM 64	—	—	—	—	—	—	—	—	—	—
DRAGON	PLAY	A, B, C, D, E, F, G	O	L	P	—	V	T	X<var>;	—
EQUIPOS MSX	PLAY	A, B, C, D, E, F, G	O	L	R	N	V	T	X <var>;	<com>=<var>;
HP-150	—	—	—	—	—	—	—	—	—	—
IBM PC	PLAY	A, B, C, D, E, F, G	O	L	P	N	V	T	X <var>;	<com>=<var>;
MPF	—	—	—	—	—	—	—	—	—	—
NCR DM-V (MS-BASIC)	—	—	—	—	—	—	—	—	—	—
NEW BRAIN	—	—	—	—	—	—	—	—	—	—
ORIC	—	—	—	—	—	—	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	—	—	—	—	—	—	—
SINCLAIR QL	—	—	—	—	—	—	—	—	—	—
SPECTRAVIDEO	PLAY	A, B, C, D, E, F, G	O	L	R	—	V	T	X <var>;	<com>=<var>;
ZX-SPECTRUM	—	—	—	—	—	—	—	—	—	—

idéntico al mencionado: basta con añadir uno o dos puntos tras la nota a alargar. El siguiente es un ejemplo válido del empleo de los puntos:

```
40 PLAY "CAC.AC..A"
```

Silencios

En la interpretación de una melodía son tan importantes los sonidos como los silencios. Un silencio, como su propio nombre indica, introduce una pausa en la que no se emite ningún sonido. Los silencios se emplean para separar notas, dando, además, el ritmo adecuado a la interpretación.

Los silencios pueden tener diferentes duraciones. La nomenclatura empleada en las duraciones de un silencio es la misma que la indicada para las notas. Así, un silencio puede durar una corchea, una negra, etc. En el macrolenguaje musical, el silencio se indica con la letra R seguido por un número. El número que acompaña a este comando adopta las mismas reglas que el parámetro asociado al comando L. A continuación, se muestra un ejemplo del uso de silencios:

Separación entre notas	
DO-RE	un tono
RE-MI	un tono
MI-FA	un semitono
FA-SOL	un tono
SOL-LA	un tono
LA-SI	un tono
SI-DO	un semitono

```
10 PLAY "CDEE"
20 PLAY "CR64DR64ER64E"
```

Las dos melodías poseen las mismas notas con idénticas duraciones. Sin embargo, el empleo de silencios en la segunda hace que las notas se interpreten ligeramente separadas. Concretamente,

las dos últimas notas parecen una sola en la primera ejecución.

Volumen y movimiento

Existen dos características que repercuten en la totalidad de la melodía a interpretar. Estas son el volumen y el movimiento. El volumen se refiere a la intensidad del sonido. Por regla general, el volumen se mantendrá fijo durante toda la emisión sonora. Esto significa que bastará con definirlo al principio.

Por otra parte, el movimiento indica la velocidad a la que se toca la melodía. Dicha velocidad marca la duración real de las notas. El movimiento viene a especificar el número de redondas por minuto.

El comando que selecciona el volumen en el macrolenguaje que se está analizando es V. Este comando irá seguido por un número que indica la magnitud de dicho volumen. La citada cantidad varía desde cero (pianissimo) hasta 15 (fortissimo). En el siguiente ejemplo se hace sonar una nota en todos los volúmenes posibles:

```
10 PLAY "V0CV1CV2CV3CV4CV5CV6CV7C"
20 PLAY "V8CV9CV10CV11CV12CV13CV14CV15C"
```

Por su parte, el movimiento se indica por medio del comando T. El número que acompaña a esta letra indicará la cantidad de negras por minuto que han de sonar. La variación que admite este comando cubre desde 32 (32 negras por minuto) hasta 255 (255 negras por minuto). De esta forma se calibra la velocidad de ejecución de la obra, desde «lento» hasta «vivace», pasando por «adagio», «andante» y «allegro».

El empleo de variables

En algunas ocasiones se emplea una misma serie de notas en distintos puntos de una melodía. En estos casos resulta tedioso repetir las mismas notas una y otra vez. El macrolenguaje musical permite almacenar esas listas de notas de forma muy flexible. Como se ha visto, los conjuntos de órdenes del macrolenguaje se manipulan en forma de cadenas de caracteres. Pues bien, también se puede hacer uso de variables de ese mismo tipo. Un posible ejemplo es el que se muestra a continuación:

```
100 LET A$="CDECDFCDG"
110 PLAY A$
■
```

Esta facilidad permite interpretar varias veces una misma melodía sin tener que teclear de nuevo toda su formulación. Pero no es ésta la única posibilidad del empleo de variables. Se pueden insertar variables en el interior de una cadena ejecutable, conteniendo éstas fragmentos de la melodía completa. Para ello se utiliza el comando X, seguido del nombre de variable de cadena y de un punto y coma. Siguiendo con el ejemplo anterior, ésta es una ampliación del mismo:

ESCALA DE SEMITONOS
DO — RE — MI FA — SOL — LA — SI DO

```
120 PLAY "O3XA$;O6XA$;"
```

En esta última línea se ha repetido dos veces el anterior fragmento. En cada una de las ejecuciones se ha utilizado una octava distinta, para diferenciarlas.

Las variables de cadena no sólo pueden contener notas, sino también indicaciones de octava, duración, etc.

Además de las variables de cadena (para listas de órdenes) se puede hacer uso de variables de tipo numérico. Estas últimas permiten almacenar los datos que se adjuntan con los comandos O, V, T, etc.: datos numéricos. Las variables se añaden tras el comando adecuado, precedidas por un signo «=» y seguidas del preceptivo punto y coma. El mismo ejemplo de más arriba se completa con la posibilidad de variar el volumen:

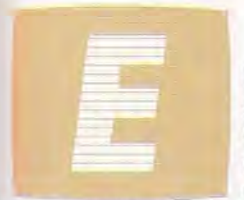
```
 10 INPUT "VOLUMEN=";VOL
100 LET A$="CDECDFCDG"
110 PLAY "V=VOL;"
120 PLAY "O3XA$;O6XA$;"
```

En este caso se recoge el lado de volumen en la instrucción INPUT de la línea 10. Ese valor se almacena en la variable VOL, variable empleada en la línea 110 para fijar el volumen.

Duraciones de las notas		
Símbolo	Duración	Selección con el comando L
redonda	duración unidad	L1 (duración unidad)
blanca	media redonda	L2 (1/2 redonda)
negra	media blanca (1/4 redonda)	L4 (1/4 redonda)
corchea	media negra (1/8 redonda)	L8 (1/8 redonda)
semicorchea	media corchea (1/16 redonda)	L16 (1/16 redonda)
fusa	media semicorchea (1/32 redonda)	L32 (1/32 redonda)
semisufsa	media fusa (1/64 redonda)	L64 (1/64 redonda)

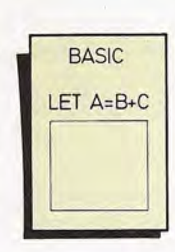
Del BASIC al código máquina

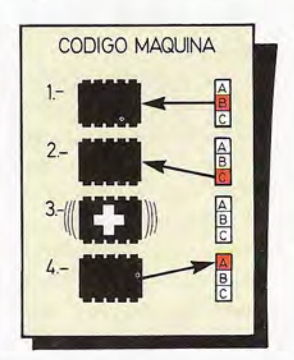
En la recóndita intimidad del ordenador



El BASIC es un lenguaje de alto nivel. Ello significa que se trata de una representación no directamente inteligible por el ordenador, sino próxima al lenguaje hablado convencional. En consecuencia, el ordenador ha de convertir las expresiones BASIC a otro código que sea capaz de entender y ejecutar: el código máquina. Así pues, ¿por qué no programar directamente en código máquina?

El manejo del código máquina tiene sus ventajas, pero también sus inconvenientes. Este es, precisamente, el tema en discusión a lo largo del presente capítulo.





Las instrucciones del código máquina son mucho más elementales que las del BASIC. Ello hace que la más simple instrucción BASIC se traduzca en varias instrucciones de código máquina.

Aspecto del código máquina

El código máquina corresponde a la representación de instrucciones más cercana al ordenador. Esta representación cabe imaginarla como filas de unos y ceros. Cada ocho de estas cifras se agrupan en un byte, y cada byte o grupo de ellos es capaz de dar cuerpo a una instrucción elemental.

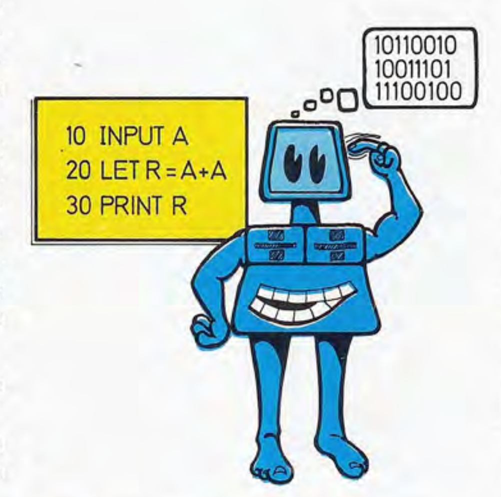
El repertorio de instrucciones depende de la configuración física (hardware) del ordenador, y más concretamente del microprocesador que constituye su cerebro. Distintos microprocesadores utilizan diferentes repertorios de instrucciones y, por lo tanto, los programas en código máquina no coincidirán.

Las instrucciones en código máquina son mucho más elementales que las del lenguaje BASIC. Por ello, una instrucción BASIC suele descomponerse en varias de código máquina. Así, por ejemplo, la actividad asociada al tratamiento de la sentencia BASIC LET A=B+C ha de seguir los siguientes pasos:

— Recoger el dato almacenado en la posición de memoria indicada como B.
— Recoger el dato almacenado en la posición de memoria referenciada como C.
— Sumar ambos datos.
— Almacenar el resultado en la posición de memoria identificada por A.

Este método sería válido si se tratara de datos de un byte (ocho bits). No obstante, si los datos a sumar fueran números enteros, se utilizarían dos bytes para codificar cada uno de ellos, con lo cual la cosa se complica. Habría que recoger dos bytes por dato y realizar dos sumas, una con cada byte. Más compleja todavía sería la suma de datos de doble precisión, que utilizan ocho bytes.

Como se observa, el código máquina puede resultar bastante incómodo a la hora de programar procesos complicados. Sin embargo, la ejecución de subrutinas en código máquina es mucho más rápida que en BASIC. Ello se debe a que el ordenador no tiene que traducir las órdenes, puesto que ya están en su lenguaje interno.



A pesar de que nos comunicamos con el ordenador en BASIC, éste no es su lenguaje «natural». La máquina se ve obligada a traducir las instrucciones BASIC a su propio código interno: el código máquina.

Empleo de código máquina en un programa BASIC

Dentro de un programa en BASIC se pueden emplear rutinas escritas en código máquina. Estas rutinas pueden coexistir perfectamente con el programa BASIC, siempre y cuando las instrucciones escritas en código máquina se sitúen en la zona de memoria adecuada.

Para escribir una rutina en código máquina hay que partir de un buen conocimiento del repertorio de instrucciones del microprocesador. Una vez planificada la rutina, ésta debe traducirse a los códigos binarios correspondientes a las respectivas instrucciones. Los números calculados se trasladan a la zona de memoria adecuada por medio del comando BASIC POKE. Este tiene como misión depositar un número en una determinada posición de memoria, y para ello hace uso de dos datos en su argumento: el primero indica la posición de memoria a rellenar, mientras que el segundo coincide con el dato a depositar en la misma. Por lo tanto, su formulación tendrá el siguiente aspecto:

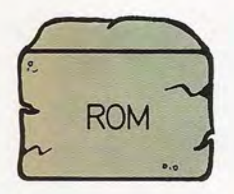
(Número de línea) POKE <dirección>, <número>

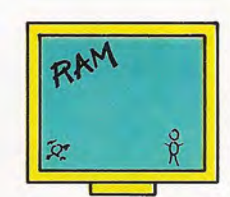
A partir del comando POKE es posible escribir un programa capaz de emplazar una rutina en código máquina en las posiciones de memoria que desee el usuario. Su intervención reiterada se consigue al incluir el comando POKE dentro de un bucle FOR/NEXT.

De la misma forma que se dispone de un comando para depositar un dato en una posición de memoria, también se puede hacer uso de una función para recuperarlo. Esta función se asocia a la palabra clave PEEK. Su formulación es la que sigue:

(Número de línea) <var.>=PEEK (<dirección>)

En este formato se ha situado a la función PEEK dentro de una sentencia de





MEMORIA

La memoria principal del ordenador consta de dos zonas bien diferenciadas: la ROM, cuyo contenido es fijo e imposible de alterar, y la RAM, en la que es posible tanto leer como escribir datos.

asignación. Sin embargo, al tratarse de una función, PEEK puede utilizarse en el argumento de un comando. Por ejemplo:

PRINT PEEK(4000)

mostrará en pantalla el contenido de la posición de memoria número 4000.

Con los comandos PEEK y POKE se puede acceder a cualquier posición de la memoria. Hay que tener en cuenta que la ROM, al ser su contenido fijo y no modificable, no permitirá el uso de POKE.

Ocupación de memoria

Cuando se desea utilizar rutinas en código máquina como complemento de un programa BASIC, es necesario controlar el espacio de memoria en uso. Ello también es de gran utilidad cuando se está introduciendo un programa muy largo y nos acercamos al límite de memoria.

El BASIC incluye una función capaz de revelar el número de bytes libres. Se trata de la función FRE. Esta admite un dato como argumento; si bien, el referido dato no afecta al funcionamiento de FRE. Por ello, suele utilizarse FRE(0) para obtener la cantidad de memoria que aún no ha sido ocupada.

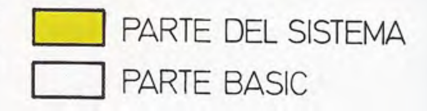
Cabe precisar que el dato proporcionado por FRE se refiere únicamente a la zona de memoria reservada a los programas BASIC; por lo que su respuesta no contempla la zona de variables BASIC, y mucho menos a los segmentos de memoria reservados al sistema.

Se ha indicado anteriormente que las rutinas en código máquina se suelen situar en la parte de memoria dedicada al BASIC. Ello exige adoptar una primera precaución: evitar que se mezclen rutinas en código máquina y programas BASIC. Al efecto, existe normalmente una variable del sistema que contiene el valor de la posición de memoria que marca el límite de la zona de programa. Este límite no puede ser rebasado durante el proceso de introducción de un programa BASIC. Así pues, es un buen método situar esas rutinas a continuación de dicho límite. Aunque ello no es siempre posible, puesto que la memoria situada a continuación puede servir para otro cometido. En tal caso, el truco consiste en «engañar» a la máquina diciéndole que la zona de programa acaba antes de lo estipulado. Para lograrlo hay que cambiar el valor de la posición límite, accediendo a la adecuada variable del sistema.

Esta misma operación puede realizarse directamente por medio de un comando BASIC. Dicho comando se invoca mediante la palabra clave CLEAR.

El comando CLEAR suele tener un doble cometido: por una parte se encarga de poner a cero todas las variables del programa, y por otra se ocupa de definir el límite anteriormente indicado. Esto





La memoria RAM se subdivide en varias zonas, de las cuales unas son utilizadas por el BASIC mientras que otras quedan reservadas para uso exclusivo del sistema.

último se consigue especificando la nueva posición límite en su argumento.

En general, la formulación de una

PEEK

Extrae el contenido de la posición de memoria indicada.

Formato: PEEK (<posición>)

Ejemplos: 20 LET D=PEEK(32500)
70 PRINT PEEK(2000)

POKE

Deposita un dato numérico en la posición de memoria especificada en su argumento.

Formato: POKE <posición>, <dato numérico>

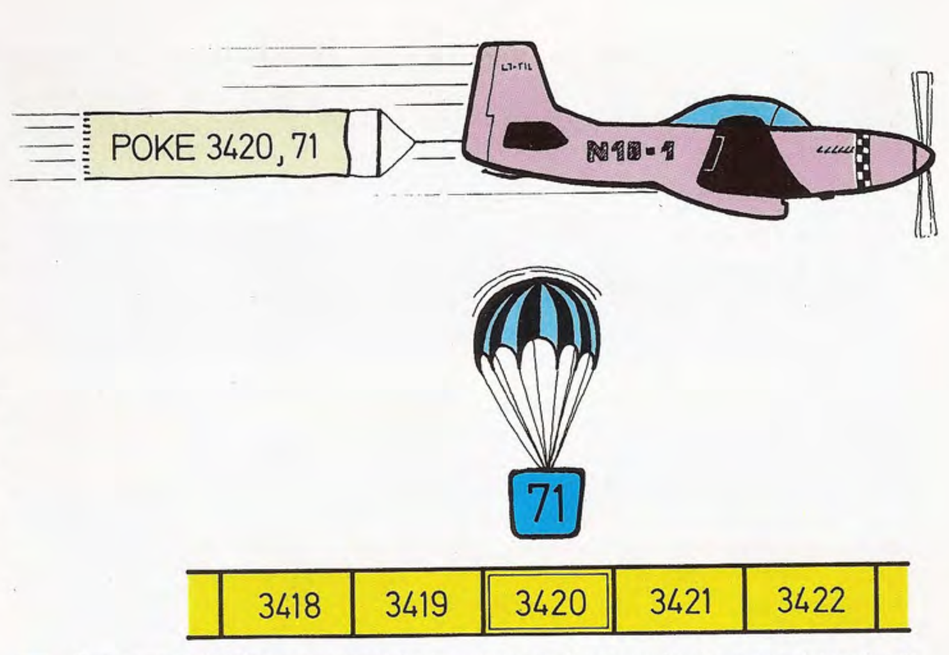
Ejemplos: 10 POKE 32500,DATO
50 POKE 2000,22

instrucción CLEAR coincide con la siguiente:

(Número de línea) CLEAR [,<posición>]

En ella, la cláusula <posición> indica la posición de memoria que se ha de tomar como límite para la zona de programa.

Al igual que suele ocurrir con muchos otros comandos BASIC, CLEAR no realiza la misma función en todos los aparatos. Cada ordenador tiene, como ya se ha señalado, una distribución de memoria diferente, lo que implica que la zona de BASIC puede estar situada en distintas posiciones. Además, es posible que el límite modificado con CLEAR no corresponda a la zona de programas, sino a la de variables BASIC.



El comando POKE se encarga de depositar un dato numérico en una determinada posición de memoria especificada en su argumento.

Utilizando rutinas en código máquina

002 Supongaqueyasehadiseñadolarutina en código máquina. Se han traducido las instrucciones a sus correspondientes códigos numéricos, y se han situado éstos en las posiciones de memoria adecuadas, mediante comandos POKE. Ahora ya sólo falta utilizarla.

El BASIC contempla dos formas de llamar a una rutina en código máquina. La primera y más elemental hace uso del comando CALL. Para emplear CALL es necesario conocer la posición de memoria en la que comienza la rutina en cuestión, posición que se ha de indicar en el argumento de CALL. Su formato muestra el siguiente aspecto:

(Número de línea) CALL <posición de memoria> [(<parámetro>[,<parámetro>, ...])]

Cuando la secuencia de ejecución del programa tropieza con un comando CALL, el control se transfiere a la rutina especificada. Al terminar la ejecución de la misma, la secuencia de proceso retorna a la instrucción BASIC situada a continuación de CALL. Se observa pues que CALL actúa como una llamada a subrutina; equivalente a GOSUB, pero con la salvedad de que, mientras GOSUB llama a una subrutina BASIC, CALL hace lo propio con una subrutina escrita en código máquina.

El comando CALL no sólo sirve para llamar a una subrutina, sino que puede realizar esa llamada «pasando un parámetro».

Si la rutina en código máquina necesita que se le proporcione dato, éste puede ser mandado, desde el programa BASIC, a modo de «parámetro». Dicho parámetro será un dato que utilizará la rutina en código máquina para elaborar sus cálculos. Es posible mandar varios

CLEAR

Borra el contenido de todas las variables del BASIC. Opcionalmente sitúa el límite de la zona de BASIC.

Formato: CLEAR [<posición>]

Ejemplos: 20 CLEAR
70 CLEAR 33450

FRE

Proporciona el número de bytes que quedan sin utilizar.

Formato: FRE(0)

Ejemplos: 20 PRINT FRE(0)
70 LET LIBRE=FRE(0)



El comando PEEK es el encargado de recuperar un dato almacenado en una posición de la memoria principal del ordenador.

parámetros, si bien el uso que se haga de ellos dependerá exclusivamente del funcionamiento de la rutina.

Existe un método alternativo para utilizar rutinas en código máquina dentro de un programa BASIC: se trata de utilizar dichas rutinas como funciones definidas por el usuario.

Funciones de usuario en código máquina

Es posible la definición de otro tipo de funciones aparte de las declaradas a través de DEF FN.

Si aquellas eran creadas bajo el ámbito del BASIC, estas nuevas funciones se escriben en código máquina, aunque el modo de operación es similar. Previamente hay que memorizar la rutina que realiza las operaciones pertinentes en el lugar apropiado. Con la rutina situada en una posición de memoria conocida,

BASIC
CLEAR
BASIC | RUTINAS EN CODIGO MAQUINA

El comando CLEAR puede ser empleado para alterar la posición límite de la zona BASIC. Con ello se reserva espacio para rutinas en código máquina.

el resto se hace directamente desde el BASIC. Para empezar, se ha de definir la función en código máquina; y como quiera que la rutina ya está creada, sólo falta indicar su dirección de comienzo. Ello se consigue utilizando el comando DEF USR cuyo formato coincide con:

(Número de línea) DEF USR<dígito>=<dirección>

En el formato se observan dos datos a especificar. Por una parte, el denominado <dígito>: dato que servirá para diferenciar unas funciones de otras. Como se trata de un sólo dígito, sólo se podrán definir diez funciones en código máquina al mismo tiempo. De todas formas, este número se revela más que suficiente en la mayor parte de los casos.

El otro dato hace referencia a la dirección de comienzo de la rutina en código máquina. Así pues, los siguientes serían ejemplos válidos de definición de funciones de esta categoría:

```
10 DEF USR1=33450
20 DEF USR3=1000
30 DEF USR6=23
```

Los ejemplos definen las funciones de usuario USR1, USR3 y USR6, cuyas rutinas en código máquina comienzan en las posiciones 33450, 1000 y 23, respectivamente. Como es lógico, en esas direcciones se deben encontrar las correspondientes rutinas.

Este es el método a seguir para definir las funciones. Para utilizarlas bastará con invocar el nombre que se ha utilizado en la definición. Al tratarse de funciones, éstas deben figurar dentro de sentencias de asignación o deben estar precedidas por comandos. Por ejemplo:

```
180 LET A=USR1(10)
200 LET B=A+USR3(A)
300 PRINT USR6(B)
```

Se observa que, al igual que ocurre con casi todas las funciones BASIC, éstas se acompañan de un parámetro situado entre paréntesis en su argumento. En realidad, las rutinas en código máquina explotadas de acuerdo a este método pueden emplear dos tipos de parámetros.

Por un lado, hay que hablar del parámetro de entrada, ya comentado. Este es idéntico al que admite opcionalmente el comando CALL. Se trata pues de un dato que se le transfiere a la rutina en códi-

CALL

Efectúa una llamada a una subrutina en código máquina situada en la posición que se indica.

Formato: CALL <posición>

Ejemplos: 10 CALL 32500

USR

Proporciona un dato calculado en una rutina escrita en código máquina.

Formato: USR<dígito>(<parámetro>)

Ejemplos: 20 PRINT USR1(35)
70 LET LIBRE=USR3(A$)

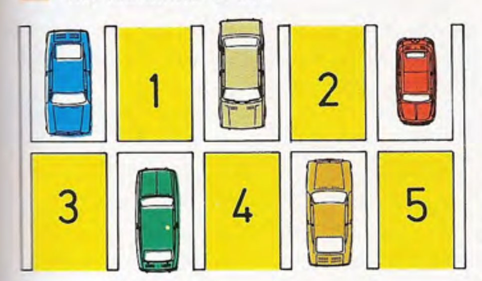


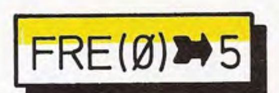






Para pasar el control de la ejecución del BASIC a una rutina de código máquina se hace uso del comando CALL.





La función FRE(0) proporciona el número de bytes libres que existen en la memoria RAM a disposición de los programas que desee introducir el usuario.

go máquina para que ésta lo procese. El otro tipo de parámetro consiste en el dato de salida proporcionado por la propia rutina; dato que se asigna a la variable (si está en una sentencia de asignación) o se muestra por pantalla (si se encuentra como argumento de un PRINT). Esta última posibilidad no la facilitaba el uso de CALL.

Todas las características relativas a los parámetros de estas funciones vienen marcadas por la correspondiente rutina en código máquina.

Almacenamiento y recuperación de rutinas en código máquina

Las rutinas en código máquina se sitúan en una zona bien determinada de la memoria. Siguiendo los pasos comentados anteriormente (uso de CLEAR para abrir espacio en la memoria), existe una diferenciación entre el programa BASIC y las rutinas en código máquina. Aparte de su distinta naturaleza, residen en zonas de memoria separadas. Ello implica que el empleo del comando NEW alterará la zona de programa, pero no la de rutinas. Para borrar estas últimas es necesario restablecer el límite inicial por medio de CLEAR.

Otro problema lo constituye la introducción de las rutinas. Esta operación se suele realizar mediante un programa «cargador», en el que se toman los códigos de la rutina de una sentencia DATA.

El referido programa puede servir para volver a crear la rutina: guardándolo en un soporte de almacenamiento externo, el programa cargador permanecerá siempre a disposición del programador.

Para volver a introducir la rutina habrá que cargar dicho programa (con LOAD) y posteriormente ejecutarlo (con RUN). Si la rutina en código máquina está asociada a un segundo programa BASIC, éste deberá cargarse a continuación.

Lo que sigue es un ejemplo general de un cargador de código máquina.

```
10 CLEAR <posición límite>
20 FOR I=<posición inicial> TO <posición final>
30 READ A
40 POKE I, A
50 NEXT I
60 DATA <códigos...>
70 NEW
```

El programa sintetiza todos los pasos necesarios para situar en memoria la

DEF USR

Define una función de usuario utilizando una rutina en código máquina.

Formato: DEF USR<dígito>=<posición>

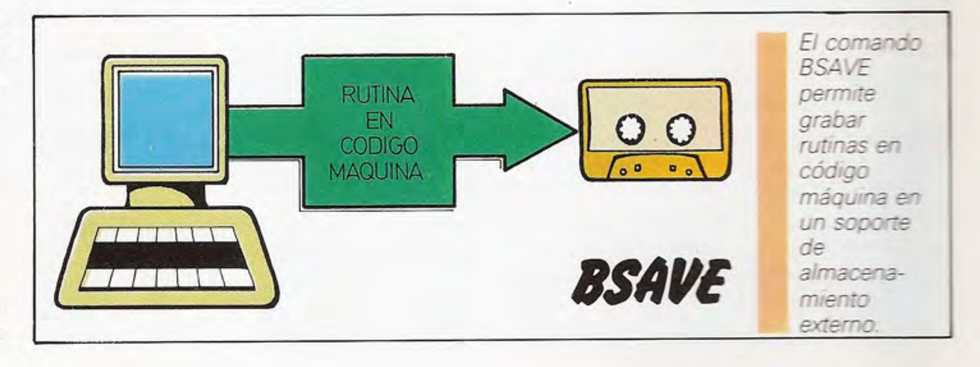
Ejemplos: 20 DEF USR3=32500
70 DEF USR7=23

BSAVE

Almacena una porción de la memoria en un soporte de almacenamiento externo.

Formato: BSAVE<nombre>,<inicio>,<bytes>

Ejemplos: 50 BSAVE "RUTINA",32500,500
85 BSAVE "CM",2000,20



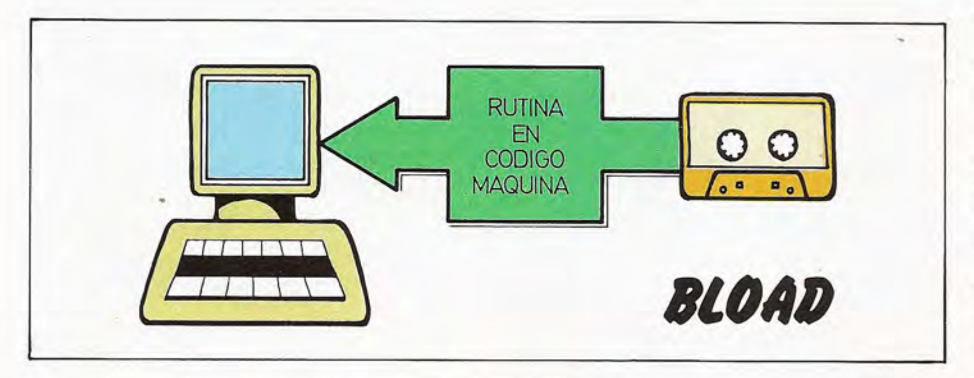
El comando BSAVE permite grabar rutinas en código máquina en un soporte de almacenamiento externo.

BLOAD

Carga en memoria una rutina en código máquina o una zona de memoria cualquiera.

Formato: BLOAD <nombre>[,<posición inicial>]

Ejemplos: 20 BLOAD "RUTINA"
70 BLOAD "CM",33000



rutina y el programa BASIC que la acompaña. La línea 10 fija el nuevo límite que reserva espacio para la rutina. Las líneas comprendidas entre la 20 y la 60 (ambas inclusive) cargan la rutina en código máquina. Por último, la línea 70 destruye el programa cargador, dejando todo listo para la introducción del programa BASIC que haga uso de la rutina creada.

Se podrían crear varios programas de este tipo, cargando en cada uno de ellos una o varias rutinas en código máquina. Estos programas son fácilmente almacenables y recuperables con los conocidos comandos SAVE y LOAD.

Sin embargo, lo idóneo sería almacenar el segmento de memoria que contiene las susodichas rutinas. Pues bien, esto es factible. Para ello han de introducirse dos nuevos comandos BASIC. El primero es un comando capaz de almacenar parte del contenido de la memoria en un dispositivo externo; se trata de BSAVE (de Binary SAVE: almacenamiento binario). BSAVE almacena cualquier porción de la memoria tratándola como una lista de bytes. Así pues, la porción de memoria estipulada se guardará byte a byte en el soporte de almacenamiento externo.

Dicho comando incluye en su argumento, junto con el nombre del archivo a crear, dos datos que especifican la zona de memoria que se almacenará. Su formato es el siguiente:

(Número de línea) BSAVE <nombre>, <dirección inicial>, <número de bytes>

Donde <nombre> debe especificar el nombre del archivo en el que va a residir la información, mientras que <dirección inicial> muestra el byte de comienzo de la zona de memoria a almacenar. El tercer dato aporta la longitud de la misma, medida en bytes.

Por ejemplo, para guardar una rutina en código máquina cuya dirección de comienzo es la 32500 y que agrupa a 150 bytes, basta con ejecutar la siguiente instrucción:

BSAVE "RUT1",32500,150

La referida información se deposita, en nuestro ejemplo, en un archivo cuyo nombre es "RUT1".

Las rutinas así almacenadas pueden ser devueltas a la memoria del ordenador por medio del comando BLOAD. El formato general de este nuevo comando es el siguiente:

(Número de línea) BLOAD <nombre> [,<dirección inicial>]

Como en el caso anterior <nombre> indica el nombre del archivo que, en este caso, se desea cargar en memoria. El segundo argumento señala, opcionalmente, la nueva dirección de comienzo. Este último dato permite ubicar la misma rutina en una posición diferente. Si no se especifica la dirección de comienzo, se cargará en el mismo lugar del que la rutina fue leída y grabada en la unidad externa con BSAVE.

Como se observa, el modo de funcionamiento de BSAVE y BLOAD es muy parecido al de SAVE y LOAD. Utilizando estos nuevos comandos para el almacenamiento y recuperación de rutinas en código máquina, se puede crear con comodidad una biblioteca de rutinas de esta categoría.

El programa que se encargaba de situar en memoria la rutina en código máquina se simplifica en gran medida con el uso de BLOAD. Este sería el aspecto del nuevo programa cargador:



La función VARPTR sirve para localizar la posición de memoria en la que se almacena el contenido de una variable dada.

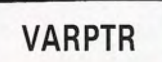
TABLA DE CONVERSION									
Ordenador	Acceso a posiciones de memoria		Espacio de memoria		Llamada a rutinas en código máquina		Almacenamiento y recuperación de rutinas en C. M.		Localización de variables
	POKE	**PEEK**	**FRE (0)**	**CLEAR**	**CALL**	**DEFUSR**	**BSAVE**	**BLOAD**	**VARPTR**
AMSTRAD	POKE	PEEK	FRE (0)	MEMORY	CALL	—	SAVE "<nom>",[1] B, <rango>	LOAD "<nom>",[2] <dir. carga>	—
APPLE II (APPLESOFT)	POKE	PEEK	FRE (0)	HIMEM	CALL/USR	—	BSAVE	BLOAD	—
APRICOT (M-BASIC)	POKE	PEEK	FRE (0)	CLEAR	CALL	DEFUSR	BSAVE	BLOAD	VARPTR
ATARI	POKE	PEEK	FRE (0)	—	USR	—	—	—	ADR (<var>)
CBM 64	POKE	PEEK	FRE (0)	—	USR	—	—	—	—
DRAGON	POKE	PEEK	MEM	CLEAR	—	DEFUSR	—	—	VARPTR
EQUIPOS MSX	POKE	PEEK	FRE (0)	CLEAR	CALL	DEFUSR	BSAVE	BLOAD	VARPTR
HP-150	POKE	PEEK	FRE (0)	CLEAR	CALL	DEFUSR	BSAVE	BLOAD	VARPTR
IBM PC	POKE	PEEK	FRE (0)	CLEAR	CALL	DEFUSR	BSAVE	BLOAD	VARPTR
MPF	POKE	PEEK	FRE (0)	Sólo borra variables	—	—	—	—	—
NCR DM-V (MS-BASIC)	POKE	PEEK	FRE (0)	CLEAR	CALL	DEFUSR	—	—	VARPTR
NEW BRAIN	POKE	PEEK	FREE	Sólo borra variables	CALL	—	—	—	—
ORIC	POKE	PEEK	FRE (0)	HIMEM	CALL	DEFUSR	—	—	—
SHARP MZ-700 (MZ-BASIC)	POKE	PEEK	SIZE	LIMIT	USR	—	—	—	—
SINCLAIR QL	POKE	PEEK	—	Sólo borra variables	CALL	—	SBYTES	LBYTES	—
SPECTRAVIDEO	POKE	PEEK	FRE (0)	CLEAR	CALL	DEFUSR	BSAVE	BLOAD	VARPTR
ZX-SPECTRUM	POKE	PEEK	—	CLEAR	USR	—	SAVE "<nom>" CODE	LOAD "<nom>" CODE	—

OBSERVACIONES:
[1] <rango> indica la zona de memoria a salvar.
[2] <dir. carga> indica la posición inicial donde se cargará el programa.

```
CLEAR <posición límite>
BLOAD <nombre del archivo>
NEW
```

Localización de variables

El modo de averiguar la posición en la que se sitúa el contenido de una variable implica el uso de una nueva función BASIC: VARPTR, cuyo formato ofrece el siguiente aspecto:



Proporciona la posición de memoria en la que se almacena el contenido de una variable.

Formato: VARPTR (<variable>)

Ejemplos:
```
20 LET A=VARPTR(BL$)
70 PRINT VARPTR(A)
```

(Número de línea) LET <variable numérica>=VARPTR (<nombre de variable>)

VARPTR es una función, lo cual significa que debe ir dentro de una sentencia de asignación, o como argumento de un comando u otra función. En el formato se ha situado dentro de una sentencia de asignación, por lo que el dato proporcionado por VARPTR se depositará en la variable que acompaña a LET.

Dentro del argumento de VARPTR se indica el nombre de la variable a localizar, esto es: de la variable cuya posición de memoria se desea conocer.

La función VARPTR también puede ser utilizada para pasar una posición de memoria como dato destinado a una rutina en código máquina. En el ejemplo que se muestra a continuación se manda como parámetro la posición en la que se almacena la variable A$:

```
150 PRINT USR5(VARPTR(A$))
```

Al igual que los restantes comandos y funciones introducidos en este capítulo, VARPTR será de especial interés para programadores experimentados.

El microprocesador: un «cerebro» electrónico integrado

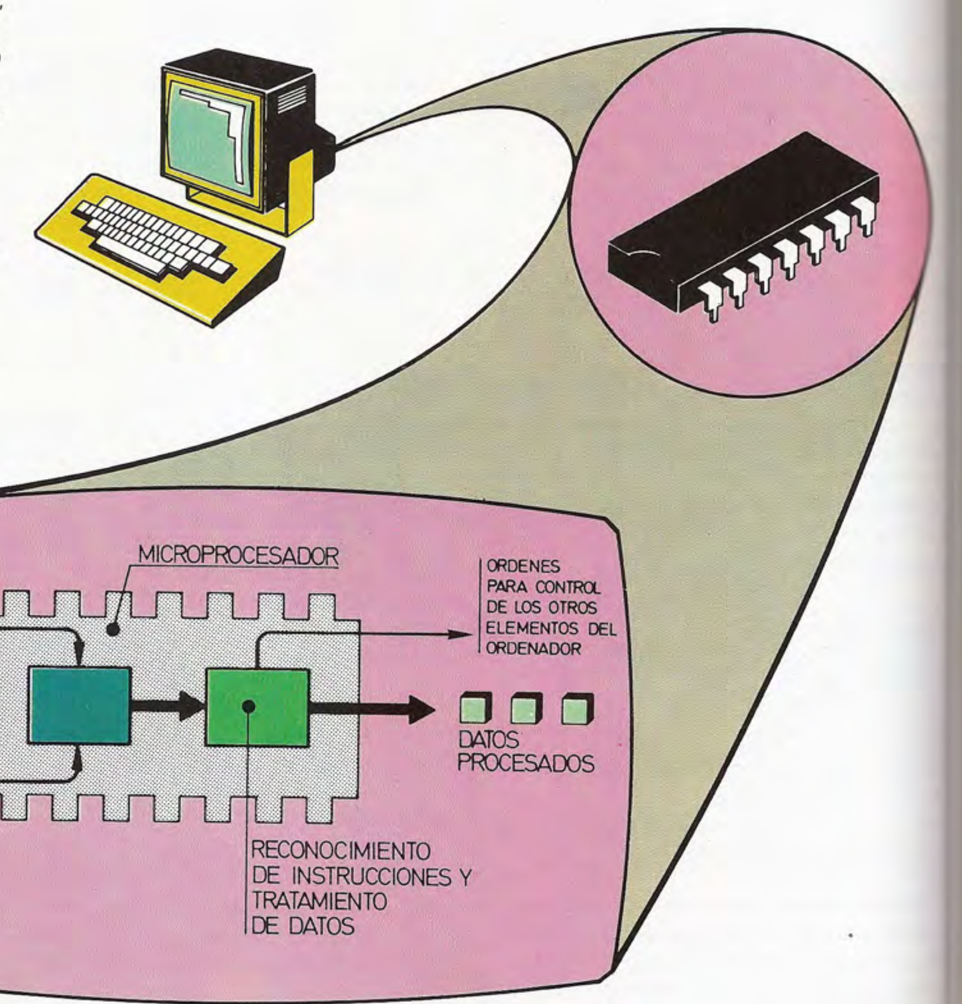
La revolución informática es un hecho cuyo origen se encuentra en el vertiginoso avance de la electrónica. Aún está reciente en el tiempo la época en que la tecnología electrónica se edificaba por completo sobre las voluminosas y frágiles válvulas de vacío. Pronto, llegaron los semiconductores, con el transistor al frente; su reducido volumen y consumo energético no tardó en relegar a las válvulas casi al olvido. El tercer peldaño lo aportó el desarrollo de la microelectrónica. Con ella se abría un nuevo horizonte de evolución, tan amplio como creciente ha sido el nivel de integración hacia el que avanzaron los circuitos electrónicos.

Hacia finales de la década de los 60, se crearon los primeros circuitos integrados LSI (Large Scale Integration), de alta escala de integración. Circuitos que alojaban cerca de 10.000 transistores o componentes electrónicos activos en una superficie inferior a un centímetro cuadrado.

El avance no se detuvo y en el mes de noviembre de 1971 irrumpió en los canales comerciales un «chip» cuya referencia era INTEL 4004: el primer microprocesador. En la actualidad —con poco más de una década por medio— se producen chips que integran más de cien mil componentes activos y han visto la luz un considerable número de microprocesadores de 8, 16 e incluso de 32 bits.

En sus orígenes, el microprocesador nació como alternativa a la multiplicidad de circuitos integrados lógicos, orientados a aplicaciones específicas, que debían fabricarse en series reducidas y de dudosa rentabilidad.

Muy pronto, su versatilidad proyectó al microprocesador en el mundo informático. Este pasó a convertirse en la unidad central de proceso de los microordenadores. Un «micro-cerebro» encargado de la manipulación y tratamiento de los datos, del control de las operaciones y de la coordinación de la actividad del conjunto de dispositivos que dan cuerpo al ordenador. Todo ello, de acuerdo a las órdenes que derivan de la ejecución de una secuencia de instrucciones denominada programa.



BASIC avanzado

Gestión de interrupciones y teclas de función

La ejecución secuencial de las líneas de un programa escrito en BASIC, no permite el control del mismo en consonancia con el tiempo real. Así, normalmente, no será posible llevar a cabo una determinada acción en el mismo momento en el que se produce la situación que da lugar a ello, sino que será necesario esperar a que el programa llegue a la línea en la que se ha codificado la detección de dicho evento, con lo que siempre se producirá un cierto retraso.

Un ejemplo puede ser el siguiente. Suponga que se tiene un ordenador conectado, mediante la interface adecuada, a un detector de la presión de una caldera de vapor. El ordenador debe controlar mediante un programa BASIC que la presión de la caldera no supere un cierto valor máximo, de peligro, y explote en consecuencia. Si se alcanza ese valor máximo, el ordenador debe dar una señal de aviso, o bien actuar sobre la caldera, mediante otra interface apropiada, para bajar la presión.

La programación de este objetivo puede ser tan simple como ordenar una lectura continua del vapor de la presión de la caldera, y comprobar si ha superado el valor máximo. Aunque bien es cierto que utilizar a todo un señor ordenador tan sólo para este cometido puede ser un gran lujo. Lo habitual es que, a la vez, se puedan ejecutar otro tipo de programas y, cada cierto intervalo de tiempo, detener la ejecución de estos para dedicar unos breves instantes al que controla la caldera.

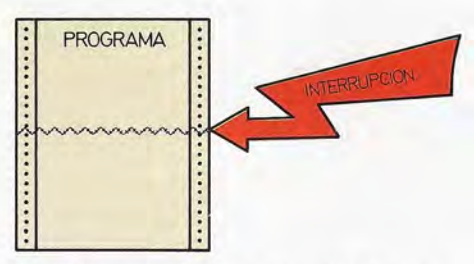
Como es fácil intuir, puede suceder que uno de los otros programas demore demasiado en su ejecución y cuando se intente examinar la presión de la caldera, sea ya demasiado tarde. El resultado es fácil de imaginar. Puesto que no se puede prever el momento preciso en el que se alcanza esa situación, se necesita alguna técnica que permita «interrumpir» al ordenador en el preciso momento en el que se produzca el hecho que debe ser atendido con prioridad.

Esta técnica no es otra que la del uso de las interrupciones hardware del microprocesador. Su explicación detallada exigiría entrar en el campo del código máquina, lo que se sale fuera del objetivo de esta obra. No obstante, sin llegar a las profundidades del hardware ni a situaciones tan drásticas como las del empleo anterior, también es posible desde el BASIC la técnica de las interrupciones.

Mediante el comando <evento> OFF se pueden desactivar las interrupciones hasta el instante en el que se ejecute la orden <evento> ON.

Interrupciones en BASIC

Existen tres situaciones habituales que permiten «interrumpir» la ejecución de un programa BASIC. Estas son las siguientes: la finalización de un determinado intervalo de tiempo, que se relaciona con la palabra clave INTERVAL; la acción sobre determinadas teclas de función especiales, anexas al teclado normal, y controladas por la palabra clave KEY, y la pulsación de la tecla STOP (normalmente junto con la tecla CONTROL), cuya palabra de control es STOP.



Las interrupciones permiten detener la ejecución de un programa para atender a una rutina de tratamiento del suceso que ha provocado la interrupción.

Cualquiera de estos eventos puede generar una interrupción instantánea del programa BASIC en curso. Cada uno de ellos necesitará entonces una rutina adecuada para el tratamiento de esa interrupción, rutina que se selecciona con el siguiente comando:

```
<NI.>ON<evento>GOSUB
  <nl.>[,<nl.>,...]
```

En ella, el <evento> puede ser cualquiera de las siguientes palabras clave: INTERVAL, KEY ó STOP.

En el caso de INTERVAL se debe especificar además el número de cincuentavos de segundo que se desea que transcurran antes de producir la interrupción. Por ejemplo, la línea:

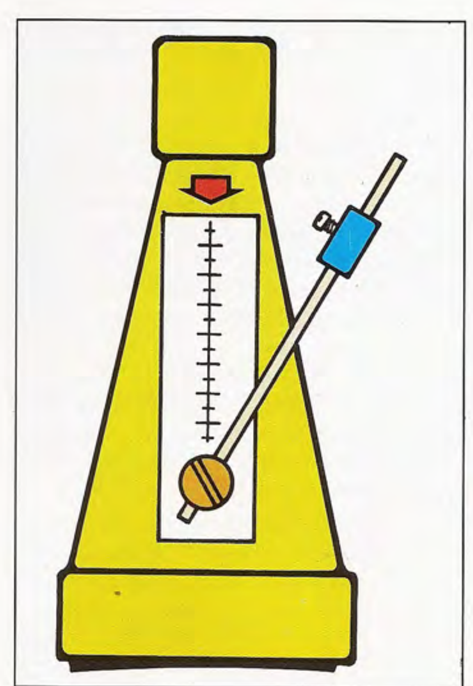
```
10 ON INTERVAL=200 GOSUB 1000
```

producirá la interrupción del programa BASIC que se esté ejecutando a intervalos de cuatro segundos, haciendo que se ejecute la subrutina de tratamiento situada en la línea 1.000.

Si se desea interrumpir el programa cuando se pulse alguna de las teclas especiales de función, se empleará la siguiente formulación, incluyendo un número de línea asociado al tratamiento de cada una de las teclas de función que existan. Así, si son cinco las teclas de función, la siguiente línea BASIC:

```
10 ON KEY GOSUB 100,200,300,,500
```

hará que se ejecute la subrutina de la línea 100 al pulsar la primera tecla, la de la línea 200 si se pulsa la segunda, y así sucesivamente. La cuarta tecla de función no producirá interrupción alguna, pues no se ha indicado el número de línea asociado a su tratamiento. No obstante, es obligatorio respetar su posición, por lo que se ha representado la «coma» correspondiente. Normalmente, existirán hasta diez teclas de función,



Con INTERVAL es posible generar una interrupción del programa principal cada vez que finalice un cierto intervalo de tiempo.

por lo que se pueden codificar otras tantas rutinas de tratamiento. Tal posibilidad resultará muy útil para facilitar el control directo de un programa por medio de ellas.

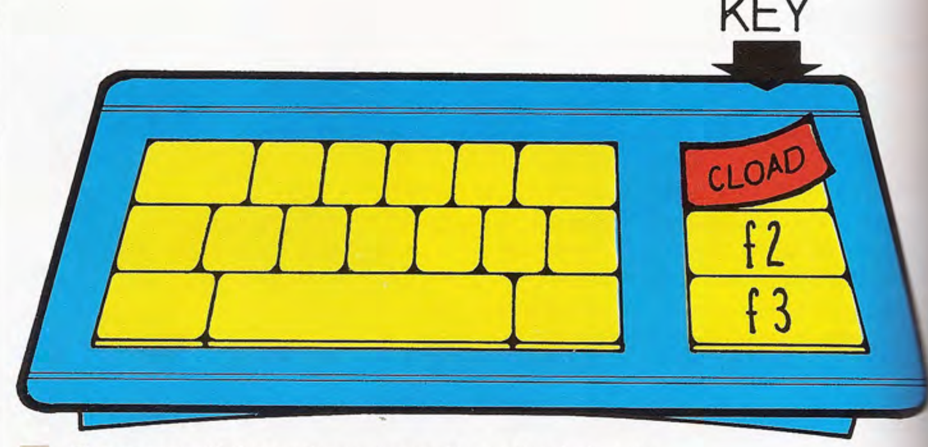
La instrucción <evento>: STOP no necesita parámetros suplementarios y será útil para interrumpir la ejecución de un programa mediante la pulsación de la tecla STOP.

En cualquiera de los casos señalados, la interrupción se produce una vez que se termina de ejecutar la sentencia BASIC en curso. Acto seguido salta la ejecución a la subrutina correspondiente y, tras ello, se retorna la instrucción siguiente a la que se estaba ejecutando cuando se produjo la interrupción.

Una vez programadas todas estas interrupciones, es posible también programar su aceptación o no en un momento determinado. Ello se debe realizar por medio de la siguiente instrucción BASIC:

<NI.><evento>[ON][OFF][STOP]

En la que <evento> puede ser uno cualquiera de INTERVAL, KEY, STOP, STRIG o SPRITE (estos dos últimos descritos en el capítulo «Control de periféricos»), e irá seguido por algunas de las palabras alternativas indicadas en el formato (ON, OFF, STOP). Previa a la ejecución de esta sentencia, debe haberse ejecutado la correspondiente instrucción ON<evento>GOSUB. Si el <evento> es KEY será preciso especificar detrás de él, y entre paréntesis, el número de tecla de función correspondiente.



El comando KEY permite programar las teclas de función, modificando sus contenidos.

La opción ON activa y permite la interrupción, de modo que ésta tendrá lugar si se produce a partir de entonces su <evento> particular. Para desactivar una interrupción, se debe elegir la opción OFF, con lo que el ordenador ignorará los eventos producidos a partir de ese momento.

La opción STOP permite desinhibir las interrupciones, pero éstas serán anotadas internamente para generarlas en cuanto se ejecute una nueva opción ON para ese mismo <evento>.

ON <evento>GOSUB

Bifurca a una subrutina al producirse una interrupción provocada por el <evento> indicado.
<eventos>: STRIG, SPRITE, STOP, KEY, INTERVAL

Formato: On <evento>: {=<num.>} GOSUB < nl.>[,...,<nl.>]

Ejemplos:
```
10 ON INTERVAL=100 GOSUB 1000
30 ON KEY GOSUB 100, 200, 300
```

Nota: <num.>=número de cincuenta-avos de sg. para INTERVAL

<evento> ON OFF/STOP

Activa o desactiva momentáneamente la interrupción asociada al <evento> indicado. Tiene relación con el comando ON <evento> GOSUB.
<eventos>: STRIG, SPRITE, STOP, KEY, INTERVAL

Formato: <evento>[ON/OFF/STOP]

Ejemplos:
```
10 KEY(3) OFF
20 INTERVAL STOP
30 STOP ON
```

Estos son algunos ejemplos que muestran la posible sintaxis de este tipo de instrucción:

```
10 KEY(2) ON
100 INTERVAL STOP
50 STOP OFF
```

La línea 10 permite poner en estado de autorización las interrupciones generadas al pulsar la tecla de función 2. La línea 100 impedirá momentáneamente las interrupciones periódicas generadas cada intervalo de tiempo indicado; éstas se producirán cuando se ejecute posteriormente una sentencia INTERVAL ON. Por último, la línea 50 hará que sean ignoradas las interrupciones generadas por medio de la tecla STOP.

Las teclas de función

A cada una de las teclas de función se le puede asignar no sólo el carácter que desee el usuario, sino incluso una cadena de hasta 16 caracteres en muchos casos, además de caracteres de control.

El comando BASIC que permite asignar una determinada cadena a una de estas teclas es el que sigue:

```
key <num. de tecla>, <cadena>
```

Donde el <núm. de tecla> es el que normalmente tiene asignado la tecla de función cuyo cometido se quiere modificar, y <cadena> es el nuevo texto a asignarle.

Así, por ejemplo, si se desea que al pulsar la tecla de función 1 se ejecute el comando RUN<CR>, es suficiente con indicar a la máquina:

```
KEY 1, "RUN"+CHR$(13)
```

En dicha orden se asume que CHR$(13) es el carácter correspondiente a la acción del retorno de carro (acción habitual de la tecla RETURN o ENTER).

Por este sencillo método se habrá asignado a la tecla de función 1 la misión de ejecutar un programa BASIC, sin que haya necesidad de escribir letra por letra el comando RUN y pulsar después la tecla RETURN. Ahora, con sólo pulsar la tecla de función 1, se ejecutará directamente el programa que resida en la memoria.

Para hacer que una de las teclas de función no ejecute nada, es decir que quede inactiva, no habrá más que asignarle la cadena nula (""), como se indica en el siguiente ejemplo:

```
KEY 10,""
```

Las posibles aplicaciones de las teclas de función son inagotables. Son útiles, por ejemplo, para imprimir textos que se repitan con cierta frecuencia, o datos como el nombre del usuario, o los comando de uso más frecuentes, como AUTO, LIST, LOAD, SAVE...

Los ordenadores que poseen estas teclas de control suelen asignarles nor-

KEY

Redefine la cadena de caracteres que se generará al pulsar las teclas de función.

Formato: KEY <núm. de tecla>, <cadena>

Ejemplos:

```
10 KEY 1, "RUN"+CHR$(13)
20 KEY 2, " "
30 KEY N, Z$
```

KEY [ON/OFF]

Borra o imprime en pantalla la línea con los contenidos de las diferentes teclas de función.

Formato: KEY [ON/OFF]

Ejemplos:

```
10 KEY OFF
50 KEY ON
```



La presencia de teclas de función programables por el usuario es un detalle frecuente en los modernos microordenadores, tanto de tipo doméstico como profesional. En la fotografía se observan las ocho teclas de función (en color azul) del microordenador ENTERPRISE.

TABLA DE CONVERSION								
Ordenador	**Interrupciones**		**Teclas de función**			**Conversión de código**		**WIDTH**
	ON <evento> GOSUB <nl.> [,...,<nl.>]	**<evento> [ON/OFF/ STOP]**	**KEY <núm.>, <var$>,**	**KEY [LIST/ON/ OFF]**	**HEXS (<núm.>)**	**OCTS (<núm.>)**	**BINS (<núm.>)**	**WIDTH (<núm.>)**
AMSTRAD	EVERY/ AFTER	DI/EI (2)	KEY <núm.>, <var$>,	—	HEXS (<núm.>)	—	BINS (<núm.>)	MODE
Apple II (APPLESOFT)	—	—	—	—	—	—	—	—
APRICOT (M-BASIC)	—	—	—	—	HEX$ (<núm.>)	OCT$ (<núm.>)	—	WIDTH [LPRINT] (<núm.>) (1)
ATARI	—	—	—	—	—	—	—	—
CBM 64	—	—	—	—	—	—	—	—
DRAGON	—	—	—	—	HEX$ (<núm.>)	—	—	—
EQUIPOS MSX	ON <evento> GOSUB <nl.> [,...,<nl.>]	<evento> [ON/OFF/ STOP]	KEY <núm.>, <var$>,	KEY [LIST/ON/ OFF]	HEX$ (<núm.>)	OCT$ (<núm.>)	BIN$ (<núm.>)	WIDTH (<núm.>)
HP-150	—	—	—	—	HEX$ (<núm.>)	OCT$ (<núm.>)	—	WIDTH [LPRINT] (<núm.>) (1)
IBM PC	ON <evento> GOSUB <nl.> (1)	<evento> [ON/OFF/ STOP] (1)	KEY <núm.>, <var$>,	KEY [LIST/ON/ OFF]	HEX$ (<núm.>)	OCT$ (<núm.>)	—	WIDTH <disp.> (<núm.>)

malmente unas tareas iniciales; éstas suelen visualizarse en una zona de la pantalla, especialmente reservada para ello, ya que suele coincidir con la línea inferior.

Debido al escaso espacio que ofrece esta línea, y a que cada tecla de función puede tener asociado un texto de hasta 16 caracteres, este texto no se visualizará completo en la línea de pantalla reservada al efecto. Para ver el contenido completo de cada tecla de función se dispone del comando KEY LIST, el cual proporciona un listado por pantalla de los contenidos íntegros de cada tecla.

Una posible ejecución de este comando daría lugar a la siguiente pantalla:

```
KEY LIST <CR>
RUN
LIST
SAVE
AUTO
SAVE "D:
LOAD "D:
CLOAD
JUAN PEREZ
SI
NO
█
```

La existencia de la línea inferior de la pantalla destinada a recordar el cometido de las teclas de función, disminuye el espacio de visualización utilizable para otras finalidades. Para eliminar estas líneas y así disponer de una pantalla limpia de mensajes se dispone del siguiente comando BASIC:

KEY [ON] [OFF]

Mediante la opción OFF se hace desaparecer de la pantalla la línea de significados de las teclas de función, mientras que para hacerla aparecer de nuevo se utilizará la opción ON.

TABLA DE CONVERSION								
Ordenador	Interrupciones		Teclas de función			Conversión de código		Width
	ON <evento> GOSUB <nl.> [,...,<nl.>]	<evento> [ON/OFF/ STOP]	KEY <núm.>, <var$>,	KEY [LIST/ON/ OFF]	HEXS (<núm.>)	OCTS (<núm.>)	BINS (<núm.>)	WIDTH (<núm.>)
MPF	—	—	—	—	—	—	—	—
NCR DM-V (MS-BASIC)	—	—	—	—	HEX$ (<núm.>)	OCT$ (<núm.>)	—	WIDT [LPRINT] (<núm.>)
NEW BRAIN	—	—	—	—	—	—	—	—
ORIC	—	—	—	—	HEX$ (<núm.>)	—	—	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	DEF KEY (<núm.>)= <var$>	KEY LIST	—	—	—
SINCLAIR QL	—	—	—	—	—	—	—	WIDTH <disp.>, (<núm.>
SPECTRA-VIDEO	ON <evento> GOSUB <nl.> [,...,<nl.>]	<evento' [ON/OFF/ STOP]	KEY (<núm.>), <var$>	KEY [LIST/ON/ OFF]	HEX$ (<núm.>)	OCT$ (<núm.>)	BIN$ (<núm.>)	WIDTH (<núm.>)
ZX-SPECTRUM	—	—	—	—	—	—	BIN	—

<evento>: cualquiera de los siguientes, STRIG, STOP, KEY, INTERVAL, SPRITE. <nl.>: número de línea. <núm.>: número entero. <var.$>: variable alfanumérica. (1) El IBM-PC admite los siguientes <eventos>: KEY(n), STRING, PEN (lápiz óptico) y COM (comunicaciones). (2) la gestión de interrupciones de los Amstrad está tratada a fondo en el capítulo del tercer tomo de BASIC dedicado al Locomotive BASIC.

<núm.>: número entero. <arch.>: nombre del archivo. <dir.1> dirección inicial. <dir.2>: dirección final. <ejec.>: dirección de ejecución. <off>: dirección offset para cargar en cualquier zona RAM. <lon>: longitud en direcciones. <disp.>: dispositivo o número de arhcivo. (2) Para impresora se usa la opción LPRINT.

Variando la anchura de la pantalla

El número de columnas visualizables en pantalla de texto varía según el modo de presentación elegido y de acuerdo a su resolución máxima. Una resolución baja no permitirá más de 20 caracteres por línea, o menos en algunos casos; mientras que una pantalla de texto de alta resolución puede llegar a contener hasta 80 caracteres por línea.

Con independencia de la resolución, que será prefijada por medio de comandos tales como SCREEN, MODE o GRAPHICS, interesa en muchos casos seleccionar a voluntad la anchura eficaz de la pantalla. De esta forma, será posible, por ejemplo, reducir su anchura y así poder disponer del espacio de pantalla restante para otros menesteres. Al efecto, algunos dialectos del BASIC inlcuyen un comando específico.

WIDTH <número de columnas>

WIDTH

Modifica el número de columnas que se visualizarán en una pantalla de texto o en una impresora. El nuevo formato de pantalla queda centrado.

Formato: WIDTH <núm. de columnas>

Ejemplos:
```
10 S CREEN 1 : WIDTH 30
30 SCREEN 1 : WIDTH 10
```

HEX$, BIN$, OCT$

Funciones que dan, a partir de un número entero, la cadena de caracteres equivalente en los sistemas de numeración hexadecimales, binario u octal, respectivamente.

Formato: <Var$>=[HEX$/BIN$/OCT$] (<núm. entero>)

Ejemplos:
```
10 A$=HEX$(100)
20 A$=BIN$(&HFFO)
40 PRINT OCT$(&B110101)
```

Su utilidad estriba en seleccionar el número de columnas que se desean visualizar para un determinado modo de pantalla de texto. El número de columnas seleccionado debe estar comprendido entre 1 y el número máximo de columnas admisible para esa resolución en modo texto.

Suponga que se trabaja con una pantalla cuya máxima resolución en modo texto es de 20 columnas por línea. Si se elige este modo y se visualiza el texto PRIMERTITULO, mediante la correspondiente orden (PRINT "PRIMERTITULO"), se obtendrá la siguiente pantalla:

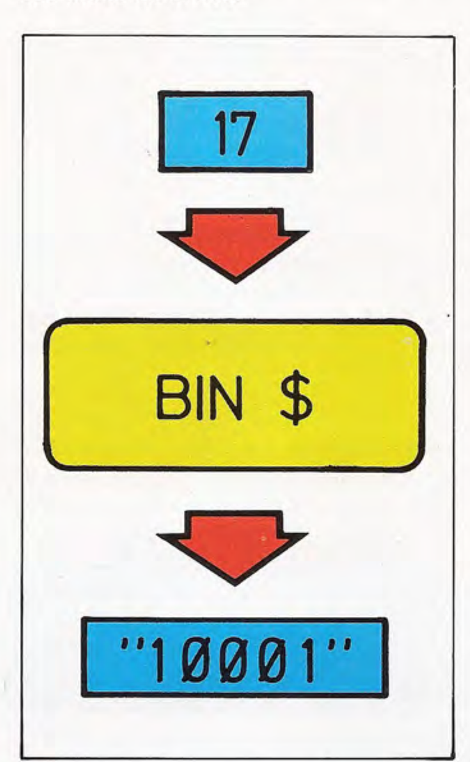
```
PRIMERTITULO
█
```

Mientras que si se ejecuta previamente a la misma orden PRINT, la instrucción SCREEN 1 :WIDTH 6 el resultado será:



Lo primero que se observa es que la anchura de la pantalla de textos ha sido reducida a 6 columnas, por lo que cada seis caracteres visualizados se produce un cambio a la línea siguiente. Además, se observa que el texto así representado queda centrado, debido a que el comando WIDTH modifica el número de columnas alternativamente a derecha e izquierda.

Una vez seleccionada la anchura, ésta quedará normalmente memorizada y en activo cada vez que se vuelva a utilizar ese modo de textos.



El BASIC dispone de la función BIN$ para transformar un número entero en su equivalente en notación binaria o base 2.

Los sistemas de numeración

Algunos intérpretes de BASIC permiten obtener los equivalentes binarios de números decimales. Como herramienta de ayuda para el manejo de números binarios, existe una función BASIC cuyo cometido es obtener una cadena de caracteres, compuesta por ceros y unos, equivalente a la expresión binaria del número decimal dado en su argumento. Esta función tiene el siguiente formato:

X$=BIN$ (<número entero>)

La conversión opuesta, es decir, el paso de binario a decimal, se puede realizar en muchos casos con la ya conocida función VAL, de la siguiente forma:

```
NUM=VAL("&B"+X$)
```

Siendo X$ la cadena de unos y ceros que representa el número en binario. Los signos &B se utilizan para indicar que se trata de un número en el sistema binario. Otro sistema de numeración, muy utilizado sobre todo por los programadores en código máquina, es el hexadecimal. Este sistema utiliza 16 dígitos diferentes para representar los números. Los dígitos se obtienen al añadir a las 10 cifras decimales (del 0 al 9), las letras de la A a la F. Existe, por tanto, otra función especial que obtiene la cadena de caracteres hexadecimales representativa del número decimal dado. Esta función tiene el siguiente formato:

X$=HEX$ (<número entero>)

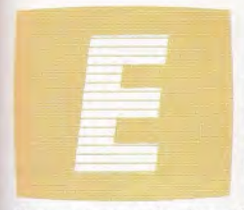
Análogamente el caso binario, para la conversión inversa se utilizará la función VAL, aunque especificando ahora que se trata de un número en notación hexadecimal, mediante los signos "&H" colocados delante del número en cuestión:

```
NUM=VAL("&H"+X$)
```

El sistema de numeración de base 8 u OCTAL, es también frecuente en el mundo informático. Para él existen también una función análoga a las BIN$ y HEX$, con las mismas características. Se trata de OCT$ cuyo formato es idéntico a los anteriores.

Presentación de datos

El comando PRINT USING



n uno de los primeros capítulos del tomo anterior se introdujo el comando PRINT. Este comando ha sido utilizado en la práctica totalidad de los capítulos de este curso de BASIC. Su importancia resulta obvia, pues casi todo proceso exige la presentación de un resultado en pantalla.

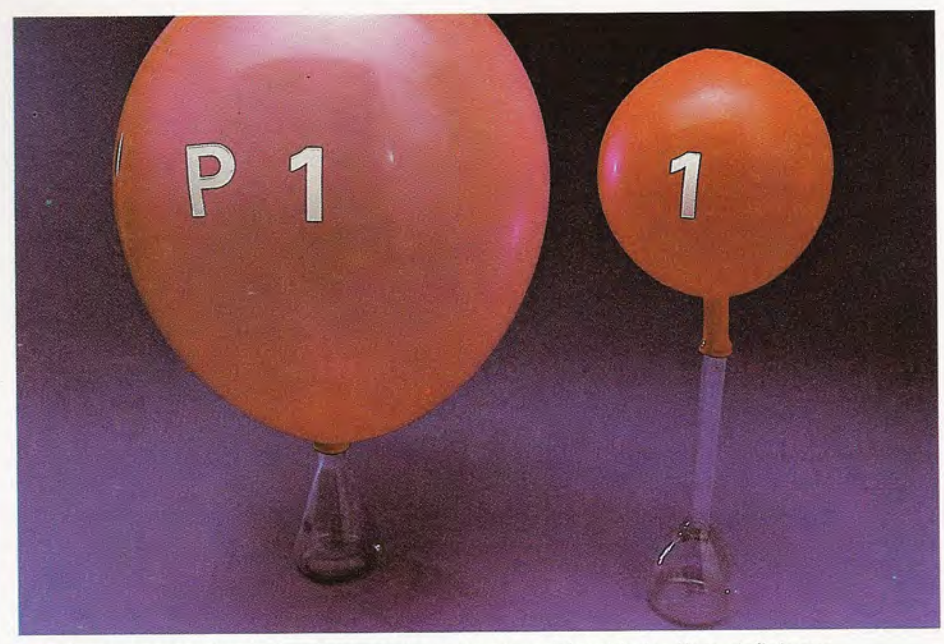
Como ya se comentó anteriormente, la presentación de datos debe ser objeto de un especial cuidado. Una presentación que adolezca de estética puede anular incluso la efectividad de un determinado programa. Para evitarlo, cabe echar mano a las facilidades que ofrece el comando PRINT.

Un ejemplo del uso del comando PRINT puede consistir en la visualización de una lista de nombres y números de teléfono. Para ello se partirá de dos matrices N$() y T(), en las que se encuentran los datos. La primera rutina encargada de mostrar en pantalla estos datos puede ser la siguiente:

```
 5 DEFDBL T
10 FOR I=1 TO 5
20 PRINT N$(I);T(I)
30 NEXT I
■
```

El resultado de la ejecución de este programa toma el siguiente aspecto:

```
RUN
MANOLO 2223344
LUIS 2341100
PEDRO 4442233
CARLOS 221122
JUAN 4005544
■
```



La presentación de los datos merece un especial cuidado. Para definir sus características de presentación, el BASIC dispone de una versátil herramienta: el comando PRINT USING.

Se comprueba que el formato de salida no es el ideal. Cabría desear una presentación en la que los números de teléfono aparecieran en la misma columna de la pantalla. Algo similar a lo siguiente:

```
MANOLO          2223344
LUIS            2341100
PEDRO           4442233
CARLOS           221122
JUAN            4005544
```



El empleo de la alternativa PRINT USING permite especificar el formato en el que serán representados los datos en la pantalla, con plena comodidad y eficacia para el programador.

PRINT USING: formatos para datos numéricos

La función USING se emplea en conjunción con el comando PRINT para especificar el formato en el que se van a presentar los datos. Dicho formato se codifica a continuación de la palabra USING, y afecta al resto de los datos contenidos en el argumento de la instrucción PRINT.

El formato se ha de indicar por medio de una cadena de caracteres que ha de ajustarse a ciertas reglas específicas.

En el caso de datos numéricos, se elige el número de dígitos a visualizar. Por cada posición se escribe un signo de número (#); como quiera que se trata de una cadena, dichos signos irán encerrados entre comillas. A título de ejemplo, se muestra una rutina que realiza la misma función comentada en el apartado anterior.

```
10 FOR I=1 TO 5
20 PRINT N$(I);TAB (10);USING
   "#########";T(I)
30 NEXT I
```

En la línea 20, la zona USING "#########" específica que se deben reservar nueve espacios (tantos como símbolos "#" haya) para la representación del siguiente dato. Si el número tiene menos cifras, se rellenan los espacios sobrantes a la izquierda con «blancos». Esta rutina daría el siguiente resultado por pantalla:



Dentro del formato no sólo se puede especificar el número de dígitos. Cuando se trata de datos numéricos con parte fraccionaria, es útil a menudo fijar el número de decimales a representar. Con el fin de reservar un espacio fijo para las zonas entera y fraccionaria, se actúa de forma muy similar a la anterior. Por ejemplo, si lo que se desea es mostrar cinco dígitos, de los cuales dos serán los correspondientes a la parte decimal, se empleará la expresión siguiente:

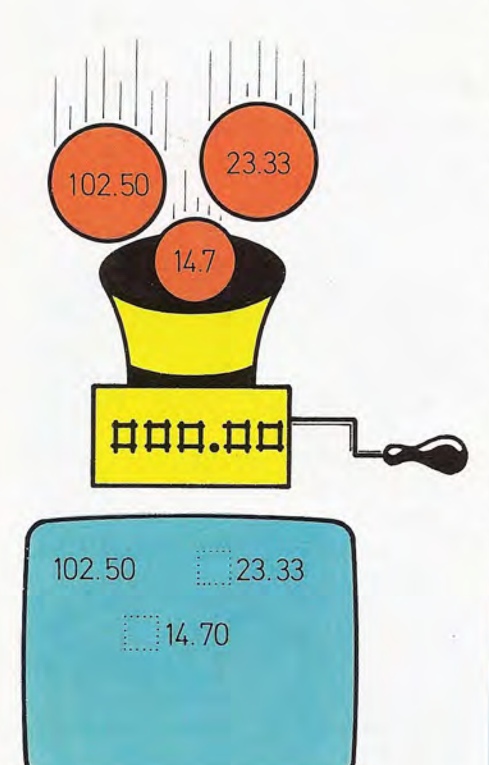
```
PRINT USING "####.##"; ...
```

El punto decimal (notación anglosajona equivalente a la coma decimal), se coloca en el lugar correspondiente, entre signos "#". Tomemos, por ejemplo, los siguientes números:

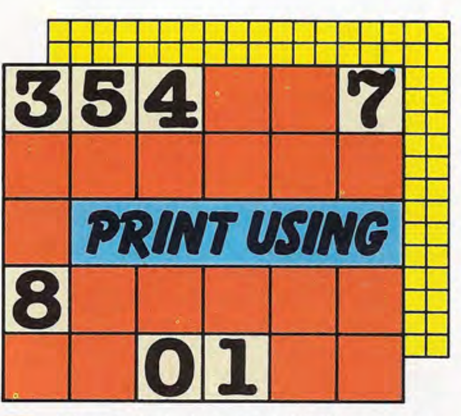
12.132 ,502.3 ,45.625 y 1208.332

Veamos qué ocurre al utilizar un formato como el anterior. El comando, ejecutado en forma directa, será el siguiente:

```
PRINT USING "###.##";
  12.132,502.3,45.625,1208.332 <CR>
12.13        502.30
45.63      %1208.33
█
```



Los signos de valor numérico (#) definen el número de espacios que hay que reservar para la presentación de los datos.



El uso de la función USING asociada al comando PRINT permite especificar el formato en el que deben ser presentados los datos.

Este ejemplo ilustra con claridad el funcionamiento del PRINT USING. En primera instancia, se reservan tantos espacios como signos "#" haya. Así, en el primer número (12.132), se deja un espacio antes del primer dígito. Ello se debe a que se han especificado 3 posiciones a la izquierda del punto decimal.

En cuanto a la parte fraccionaria, sólo se han impuesto dos dígitos; por lo tanto, sólo se representan las dos primeras cifras, truncándose las restantes. En la representación del tercer número (45.625) se ve que se redondean los decimales cuando hay truncamiento por la derecha.

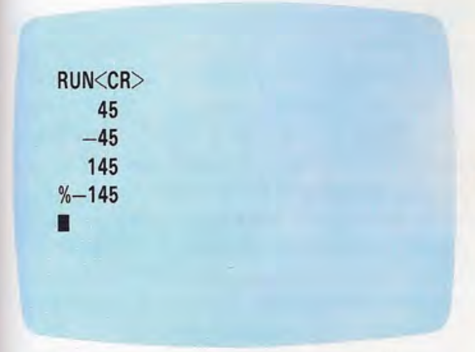
Si el número a representar excede a la cantidad de posiciones especificadas, pueden ocurrir dos cosas. Pueden faltar posiciones a la derecha del punto decimal; en tal caso no se representan los dígitos sobrantes, redondeándose el último decimal. Si, por el contrario, los dígitos sobrantes se encuentran a la izquierda del punto, estos no se truncan. Si se truncara el número por la izquierda el error cometido sería muy grande, cosa que no sucede al redondear los decimales. Para avisar de la existencia de «desbordamiento», el BASIC imprime un signo de tanto por ciento en la primera posición de la presentación. Un ejemplo de ello lo constituye el último número visualizado en el ejemplo anterior.

Formatos con signo

En el tipo de formato numérico anteriormente comentado, no se ha tenido en cuenta el signo (+ o –) del dato. Cuando el número es negativo, el signo (–) se representa consumiendo una de las posiciones reservadas. Véase el siguiente ejemplo:

```
10 PRINT USING "###";45
20 PRINT USING "###";—45
30 PRINT USING "###";145
40 PRINT USING "###"—145
```

La ejecución de estas cuatro líneas muestra cómo el signo ocupa una de las posiciones que se reservan en el formato. Este sería el resultado:



Así pues, cuando se prevean números negativos, es aconsejable dejar una posición adicional para la representación del signo. El signo positivo, en cambio, no se representa. Si se desean visualizar los números con su respectivo signo (tanto positivo como negativo) habrá que especificar tal circunstancia en el formato. Ello se consigue incluyendo un carácter «+» en el mismo. En el caso del ejemplo precedente, esta exigencia se resolvería de la siguiente forma:

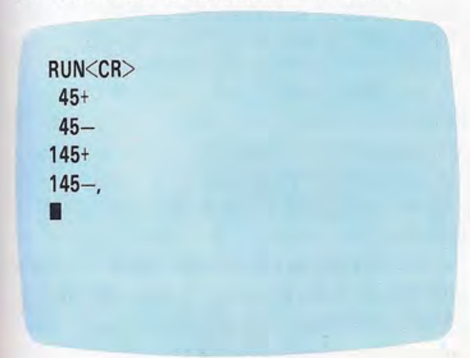
```
10 PRINT USING "+###";45
20 PRINT USING "+###";–45
30 PRINT USING "+###";145
40 PRINT USING "+###";–145
```

Ahora todos los datos se representarán con su correspondiente signo.

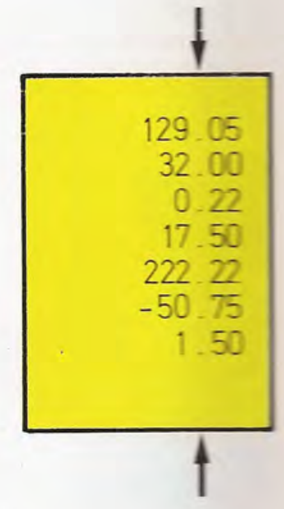
El signo (positivo o negativo) puede ser situado tanto a la izquierda como a la derecha del dato. Para ello basta con colocarlo, dentro del formato, en la posición adecuada. A continuación, se muestra el mismo ejemplo pero con el signo a la derecha.

```
10 PRINT USING "###+";45
20 PRINT USING "###+";–45
30 PRINT USING "###+";145
40 PRINT USING "###+";–145
```

Con este nuevo formato se obtendrá por pantalla el siguiente resultado:







El uso de PRINT USING facilita la representación de listas de números. Todos ellos aparecerán con el punto decimal situado en la misma columna de la pantalla.

Otros símbolos en el formato numérico

Además de la representación del signo, existen otras marcas útiles a la hora de visualizar datos numéricos. Una de éstas es la coma, que en la notación anglosajona sirve para agrupar dígitos.

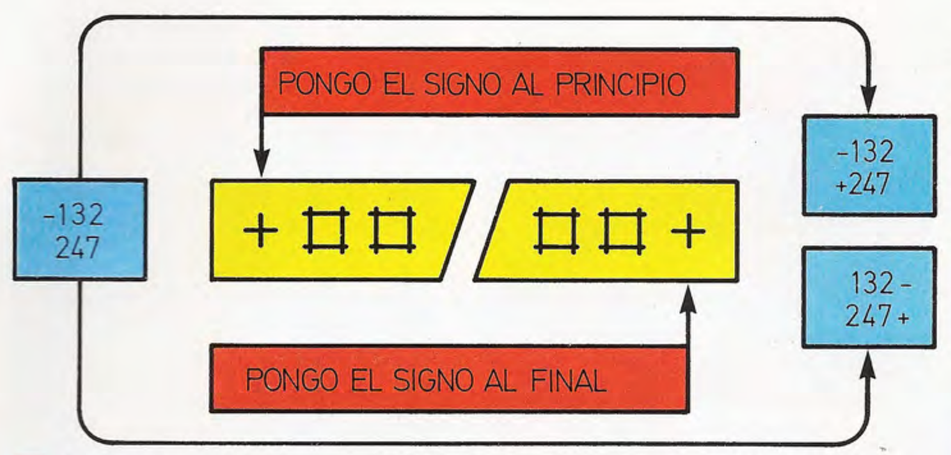
Como ya es sabido, en nuestro país (y en Europa en general) se acostumbra a separar la parte fraccionaria de la entera con una coma (2,5 en lugar de 2.5). De la misma forma, se suelen separar las unidades de millar de las centenas con un punto (1.000 representa lo mimo que 1000: mil). La notación anglosajona, por el contrario, emplea el punto para separar los decimales y la coma para agrupar los dígitos. El lenguaje BASIC, por su origen anglosajón, adopta la segunda alternativa. En la cadena de formato que sigue al PRINT USING, se puede indicar la agrupación de dígitos con comas (notación anglosajona, como ya se ha indicado). Para ello, el carácter a incluir en dicho formato es una coma, situada en cualquier posición a la izquierda del punto decimal. Véase el siguiente ejemplo:

```
10 PRINT USING "####.#";2222
20 PRINT USING "###,#.#";2222
```

En la primera línea no se ha incluido el carácter indicador de agrupamiento, mientras que la línea 20 sí lleva en su interior dicho carácter (la posición de éste podría haber sido cualquier otra *a la izquierda* del punto decimal). El resultado de su ejecución revela el efecto producido por la presencia del carácter «coma».

```
RUN<CR>
2222.0
2,222.0
```

Hay otros dos caracteres que producen un efecto específico en la representación de números. El primero es el signo de dólar ($). Dos de estos signos colocados a la izquierda del formato, harán que aparezca dicho carácter junto al dato representado. Este se visualizará inmediatamente a la izquierda del primer dígito.



La presencia del carácter «+» en el formato hará que se imprima el signo del dato; éste puede situarse tanto al principio como al final del mismo.

```
PRINT USING "$$###";289<CR>
 $289
PRINT USING "$$###";7<CR>
   $7
█
```

El otro carácter especial es el asterisco (*). Una pareja de asteriscos colocada a la izquierda, producirá el efecto de rellenar los espacios en blanco con dicho carácter. Los espacios a rellenar son los que sobren al aplicar el formato especificado.

```
PRINT USING "**###";289<CR>
**289
PRINT USING "**###";7<CR>
****7
PRINT USING "**###";1237<CR>
*1237
█
```

Como se puede observar, estos dos caracteres, al igual que el de signo, reservan también espacio para dígitos.

Se puede mezclar la acción de estos dos caracteres, logrando el efecto que se muestra en los siguientes ejemplos.

```
PRINT USING "**$##";289<CR>
*$289
PRINT USING "**$##";7<CR>
***$7
PRINT USING "**$##";1237<CR>
$1237
█
```

% 1985

ESTE DATO NO CABE EN EL FORMATO

El signo de tanto por ciento (%) aparece en pantalla dando a conocer que el dato excede el número de posiciones reservadas para su visualización.

Para representar números de magnitud extrema, muy grandes o muy pequeños, se suele echar mano de la representación exponencial. En el formato del PRINT USING se puede especificar dicha modalidad sin más que colocar cuatro signos de exponenciación (^^^^) a la derecha de los de reserva de dígitos. Estos cuatro caracteres sirven para dejar otras tantas posiciones: una para el carácter «E», una para el signo y dos para los dígitos del exponente.

```
PRINT USING "#.#^^^^";1237<CR>
1.2E+03
█
```

Todos los caracteres de indicación de formato pueden ser mezclados con entera libertad. La única salvedad está en los de representación de signo: en un formato sólo puede estar presente uno de estos caracteres.

Formateado de datos no numéricos

A la hora de representar datos alfanuméricos o de cadena de caracteres también puede especificarse un formato con PRINT USING. Se puede indicar el número de caracteres a representar, al igual que las posiciones de los dígitos en datos numéricos. Aquí no se utiliza el signo de número (#), sino que se recurre a espacios en blanco. Dichos espacios deben ir escoltados por «barras invertidas» (\), signo éste que no debe confundirse con el de división (/).

De la misma forma que los caracteres anteriores, éstos reservan dos posiciones más. Es decir, la cadena de formato «\ \» (<barra><espacio><barra>) indica que se van a representar *tres* caracteres. Estos caracteres serán los primeros (los de la izquierda) de la cadena a visualizar. Si la longitud de la cadena es menor que el espacio reservado, se rellenan los espacios sobrantes (a la derecha) con blancos. Véase un ejemplo:

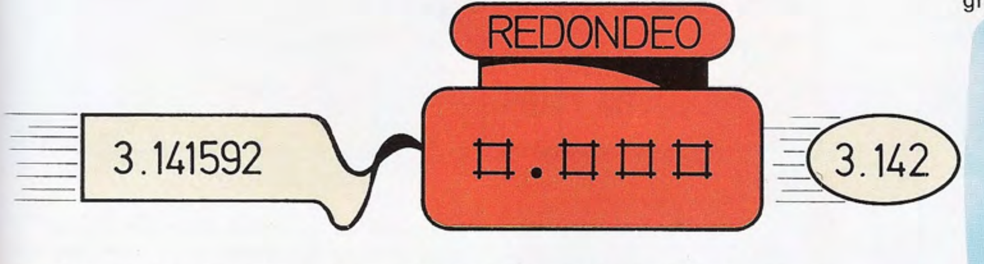
TABLA DE CONVERSION						
Ordenador		CADENAS		NUMEROS		
	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
AMSTRAD	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
APPLE II (APPLESOFT)	—	—	—	—	—	—
APRICOT (M-BASIC)	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
ATARI	—	—	—	—	—	—
CBM 64	—	—	—	—	—	—
DRAGON	PRINT USING	"%<blancos>%"	—	—	$$	**$
EQUIPOS MSX	PRINT USING	"\<blancos>\"	"&"	, (coma)	££	**££
HP-150	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
IBM PC	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
MPF	—	—	—	—	—	—
NCR DM-V (MS-BASIC)	PRINT USING	"\<blancos>\"	"&"	, (coma)	$$	**$
NEW BRAIN	—	—	—	—	—	—
ORIC	—	—	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	PRINT USING	"&<blancos>&"	—	[1]	££	—
SINCLAIR QL	—	—	—	—	—	—
SPECTRAVIDEO	PRINT USING	"\<blancos>\"	—	, (coma)	$$	**$
ZX-SPECTRUM	—	—	—	—	—	—

[1] Las comas deben situarse en el lugar real, es decir, cada tres dígitos.

```
10 A$="MI":B$="PERRO"
20 PRINT USING "\\";A$;B$
30 PRINT USING "    ";A$;B$
```

En la línea 20 no se ha dejado ningún blanco entre las dos barras inclinadas, luego se reservan dos espacios. La línea 20 incluye dos espacios entre barras, ello especifica la presentación de cuatro caracteres. El siguiente sería el resultado de ejecutar este pequeño programa:

```
RUN<CR>
MIPE
MI PERRO
■
```



Los números cuya parte decimal contiene más dígitos de los especificados en el formato son truncados por la máquina. El último dígito visualizado se redondea por la acción de este truncamiento.

La intimidad del microprocesador

En el interior de un ordenador habita una densa amalgama de componentes electrónicos que son los responsables de su correcto funcionamiento; componentes que conforman los circuitos que dan vida al ordenador. Entre ellos destaca una zona fundamental, denominada *unidad central de proceso,* la CPU (Central Processing Unit). Ella es la encargada de realizar las tareas que implican una cierta «inteligencia». Un ordenador consta de otras zonas, si bien, la fundamental es la CPU: su centro neurálgico. En los ordenadores personales y, en general, en cualquier microordenador, la CPU está condensada en un circuito integrado denominado microprocesador. Los dos cometidos principales de esta unidad son el control del sistema conjunto y la realización de operaciones. Dentro del primer cometido se encuadran las tareas de organización y gestión de la memoria y los dispositivos periféricos. Sólo la detección y reconocimiento de la pulsación de una tecla implica ya un proceso complejo, en la que la CPU necesita realizar varias operaciones de transmisión de datos.

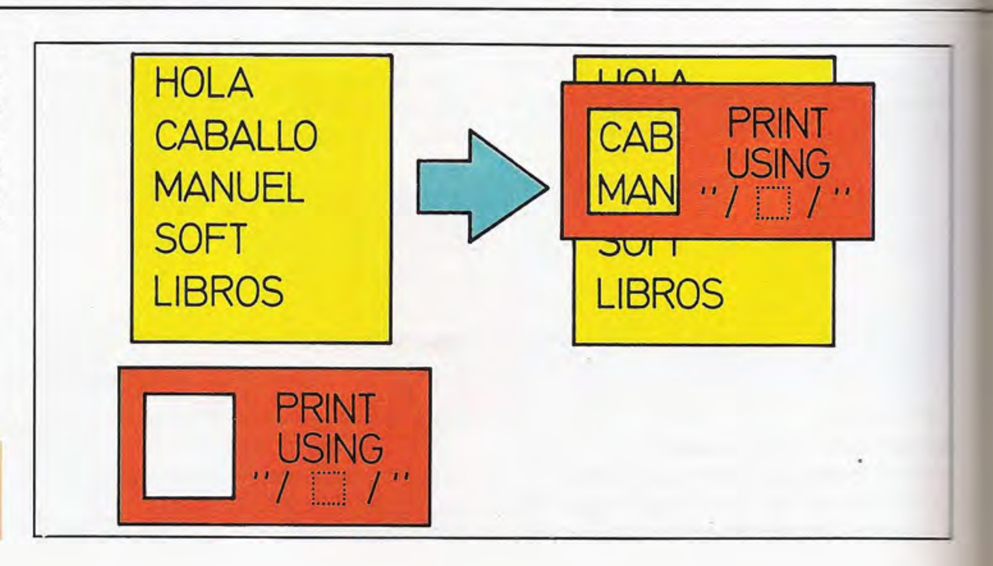
La segunda misión fundamental se concreta en el cálculo de nuevos datos a partir de los disponibles. Algo semejante a realizar la función de «calculadora» dentro del ordenador. Sin lugar a dudas, ambas actividades están mutuamente relacionadas: para sumar dos datos, por ejemplo, es necesario localizarlos en memoria, extraerlos de la misma, y hacerlos llegar al circuito sumador. A menudo suelen separarse los circuitos de la CPU dedicados a estos dos cometidos. La zona que resuelve las operaciones con los datos recibe el nombre de *unidad aritmético-lógica* (UAL). Esta, por lo general, sólo es capaz de sumar, restar y realizar algunas operaciones lógicas (AND y OR). Las operaciones se efectúan sobre los datos contenidos en registros especiales y el resultado se guarda en un registro específico denominado *acumulador.* La *unidad de control* (UC) es la otra zona básica de la CPU. Tal como su nombre indica, ésta es la que se encarga de controlar y supervisar la actividad global del ordenador. Se ocupa de hacer llegar los datos a los registros correspondientes, de enviar el resultado del acumulador a la zona de memoria adecuada y, en general, de trazar el ritmo de trabajo de la máquina.

La característica más importante de un ordenador radica en su aptitud para recibir una programación. Ello corre a cargo de una secuencia de órdenes —el programa— que definen con todo detalle las tareas que debe realizar. Las órdenes se codifican de forma inteligible para el aparato y se guardan en la memoria del mismo.

Para ejecutar un programa, la CPU ha de realizar múltiples operaciones elementales. En primer lugar, ha de «leer» la referida lista de instrucciones. Para ello empezará extrayendo de la memoria el primer dato coincidente con la primera instrucción codificada. La CPU dispone de capacidad suficiente para interpretarla. Tal actividad se apoya en el denominado decodificador de instrucciones, que forma parte de la unidad de control. Una vez «asimilada» la orden, hay que planificar la secuencia de acciones elementales que darán pie a su ejecución.

Así, por ejemplo, cursar la orden BASIC LET A=B+C, exige a la unidad de control acometer los siguientes pasos:

Se puede apreciar que con este método, el número mínimo de caracteres a exhibir es de dos (uno por barra). Para representar un solo carácter se ha de variar el formato. En este caso se pone simplemente un signo de admiración (!), éste indica que se mostrará tan sólo la inicial del dato.



Al formatear datos alfanuméricos por medio de PRINT USING se puede indicar el número de caracteres que hay que visualizar.

1. Localizar la zona de memoria donde se encuentran los datos A y B.
2. Transmitir dichos valores, tras leerlos en la memoria, a los registros de la unidad aritmético-lógica (UAL).
3. Ordena a la UAL que opere la suma de ambos datos.
4. Recoger el resultado del acumulador, y llevarlo a la zona de memoria reservada a la variable A.

El orden de las sucesivas tareas elementales es estricto; no se puede realizar la suma sin antes disponer de los sumandos. Para ello, la CPU dispone de una especie de reloj interno que le permite encadenar los diversos pasos o microórdenes.

La unidad de control ha de supervisar todo el proceso y conocer en cada instante qué tareas se han realizado y cuáles quedan por cursar. Asimismo, debe saber cuál es la siguiente instrucción a leer y ejecutar.

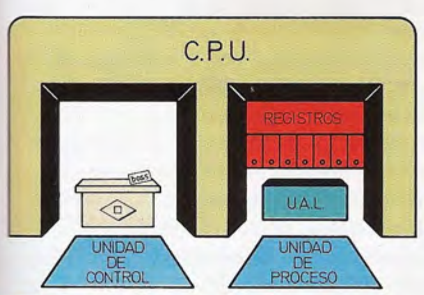
Otro de los elementos primordiales de la CPU es el *contador de programas*. Se trata de un registro que indica en todo momento cuál es la dirección de memoria en la que está almacenada la próxima instrucción a ejecutar.

En síntesis, la unidad de control consta de los siguientes elementos básicos:

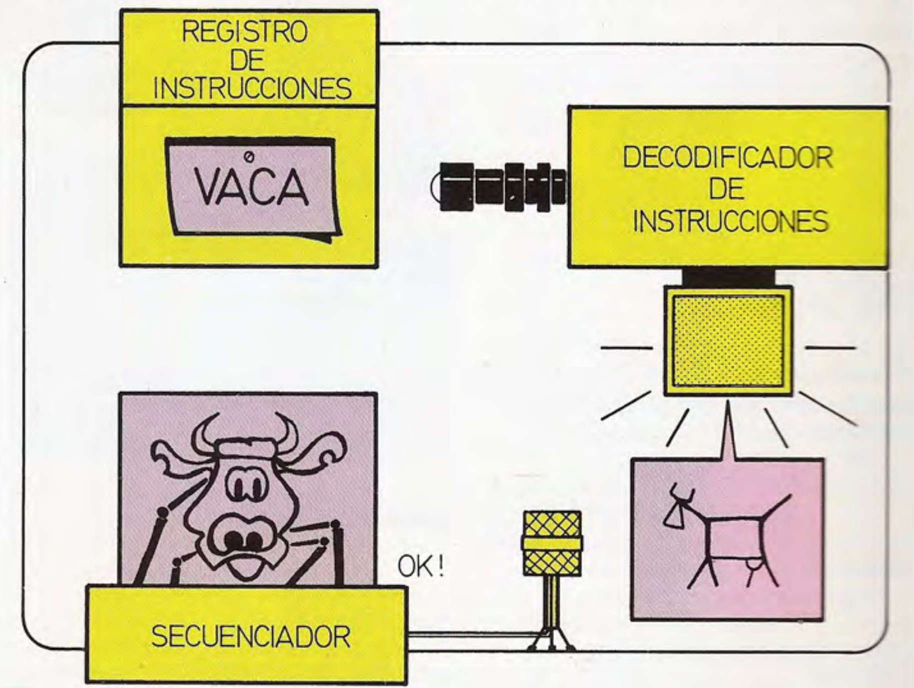
Registro de instrucciones, en el que se guarda la instrucción en curso. La CPU la traslada de la memoria a este registro para «tenerla más a mano». Aquí es donde puede proceder a su identificación.

Decodificador de instrucciones. Se trata de un circuito capaz de «reconocer» la instrucción en curso. Este inspecciona el contenido del registro de instrucciones y determina de qué orden se trata.

Secuenciador. Este circuito es el encargado de dar curso a la acción identificada por el decodificador. Una vez que éste ha interpretado la instrucción recibida, entra en actividad el secuenciador, quien se responsabiliza de ejecutarla. Para ello genera las microórdenes adecuadas (trapaso de datos, operación de la UAL...).



Dentro del microprocesador cabe diferenciar dos zonas básicas: la unidad de control y el bloque de proceso. Este último está constituido por la unidad aritmético-lógica y una serie de registros de trabajo.



Los elementos básicos de la unidad de control son el registro de instrucciones (almacena la orden en curso), el decodificador de instrucciones (interpreta y decodifica el significado de la orden) y el secuenciador (encargado de activar las acciones necesarias para su ejecución).

```
20 PRINT USING "!";A$;B$
RUN<CR>
MP
MI PERRO
█
```

Por último, se dispone de un indicador adicional para visualizar el dato en toda su longitud. Se trata del signo «&». La utilización de este signo equivale a suprimir la incidencia de la partícula USING en el argumento de PRINT.

A diferencia de lo que ocurre con los datos numéricos, aquí no está permitido unir varios indicadores de formato. La razón es muy simple: especifican acciones incompatibles.

Uso avanzado de PRINT USING

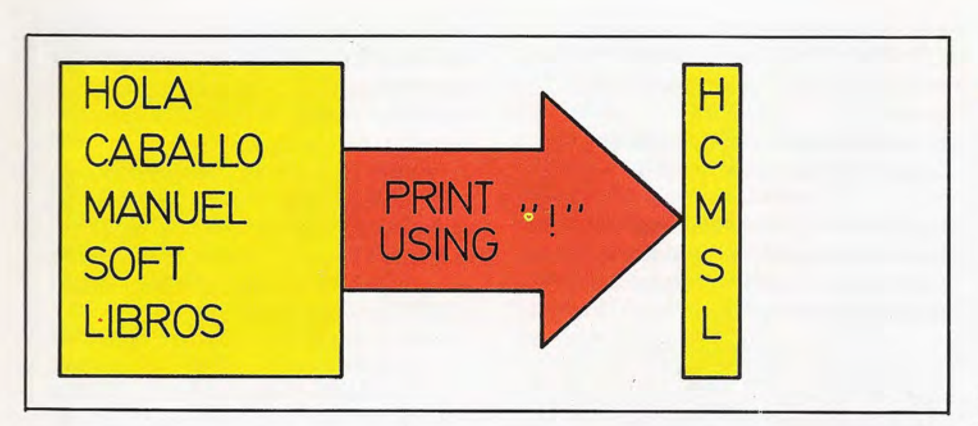
El formato con el que se va a presentar el dato se especifica por medio de una cadena de caracteres especiales.

PRINT USING

Presenta en pantalla los datos usando el formato especificado como una cadena de caracteres.

Formato: PRINT USING <formato>; <dato 1>;[<dato2>...]

Ejemplos: 20 PRINT USING "###.#"; 127.45
70 PRINT USING "!";A$;B$



El uso del signo «!» hará que sólo se visualice el primer carácter de la cadena especificada.

Esta cadena no tiene, necesariamente, por qué ser un dato fijo. Puede utilizarse una variable para determinar el formato. El siguiente es un ejemplo de formato variable.

```
10 LET A$="**#.##"
20 PRINT USING A$;10
```

En él se hace uso de la variable de cadena A$ para almacenar el formato de presentación. Este es el resultado de ejecutar las dos líneas precedentes:

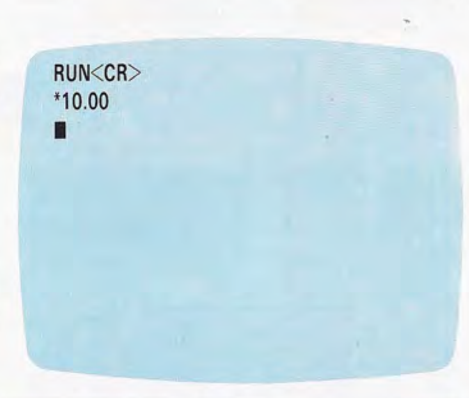


TABLA DE OPCIONES DEL COMANDO PRINT USING	
Cadenas	
!	Se muestra únicamente la inicial del dato de cadena.
\<blancos>\	Se representan tantos caracteres como blancos haya más dos. Si la cadena no tiene ese número de caracteres se justifica a la izquierda.
&	Se representa la cadena tal como es.
Números	
#	Se utiliza para especificar el número de dígitos a reservar.
.	Indica el lugar en el que se ha de situar el punto decimal.
+	Se presenta el signo del número, a la izquierda o a la derecha de éste.
—	Colocado a derecha exhibe el signo de los datos negativos.
**	Rellena los espacios en blanco, por la izquierda, con asteriscos.
$$	Muestra un signo «$» a la izquierda del primer dígito.
**$	Muestra un signo «$» a la izquierda del primer dígito, rellenando los espacios de más a la izquierda con asteriscos.
,	Sitúa comas cada tres dígitos a la izquierda del punto decimal.
^^^^	Fuerza la representación exponencial del dato.

Utilizando variables para el almacenamiento de formatos, es posible que sea el propio programa quien elija el formato, de entre los almacenados en variables. También pueden generar distintos formatos empleando las funciones para el tratamiento de cadenas.

En el formato pueden incluirse caracteres distintos de los comentados. Estos no producirán más efecto que el de aparecer en pantalla. Un posible formato, conteniendo caracteres no estándar, es el que muestra la siguiente pantalla:

```
10 PRINT USING "NUMERO**###";99
RUN<CR>
NUMERO***99
```

También es posible especificar formatos para más de un dato, aunque sean de distinto tipo. En el siguiente ejemplo se crea un formato para un dato de cadena y otro de tipo numérico.

```
10 PRINT USING "\ \**###;"NUMERO";99;
   "TE";121212
RUM<CR>

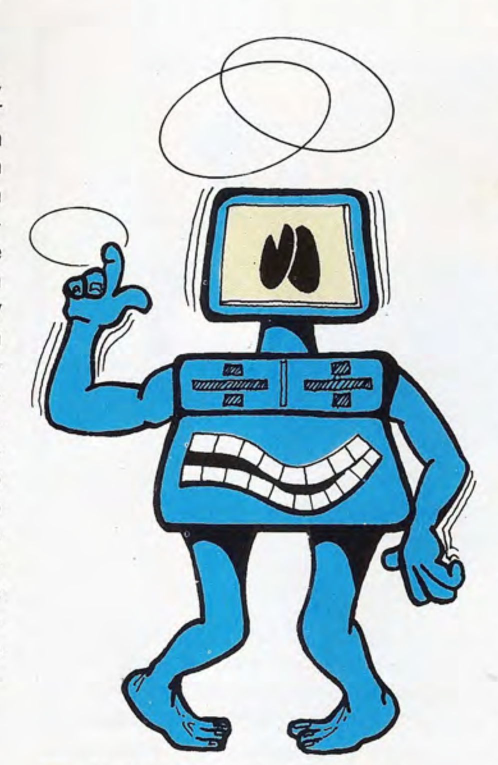
NUME***99TE %121212
```

En este último caso, es obligado situar los datos en el orden correcto; de otro modo, se produciría un error.

Tratamiento de errores

Detección y supresión de errores en los programas BASIC

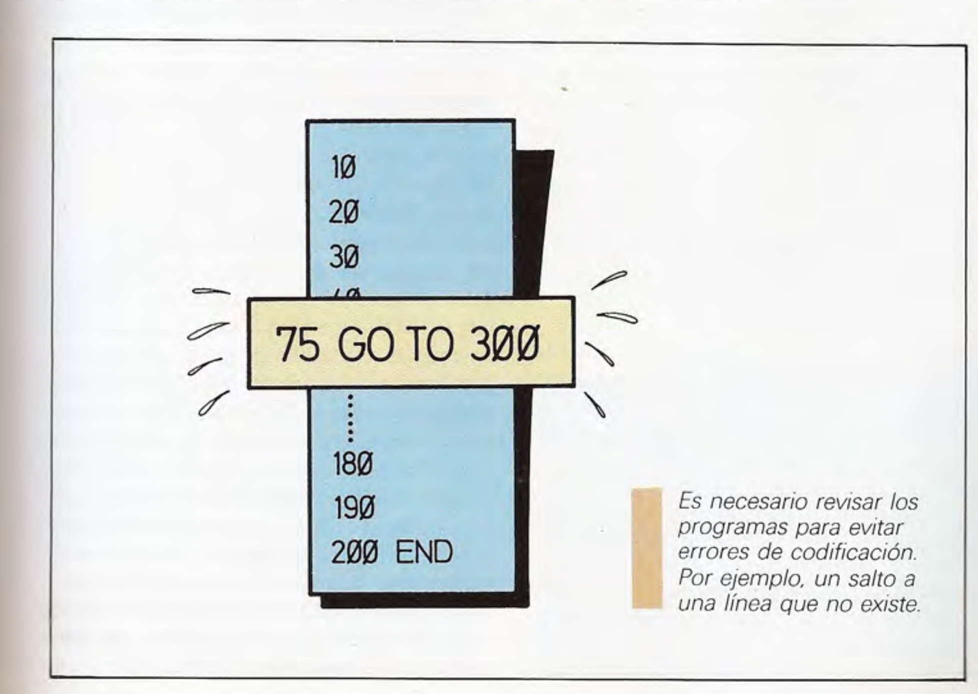
Tarde o temprano, todo programador o usuario tropieza con un programa que no funciona debidamente. ¿Qué le ocurre al programa? En principio, cabe pensar que debe tratarse de un error cometido durante la introducción del mismo. Así pues, hay que localizar y subsanar ese error. Esta tarea no siempre resulta fácil. Habitualmente, es necesario disponer de un listado por impresora, y repasarlo a conciencia. La dificultad se ve incrementada cuando el programa ha sido escrito por otra persona. En tal caso, es preciso comprender previamente el funcionamiento del mismo. A lo largo de este capítulo se comentan algunos métodos encaminados a la detección y eliminación de errores, así como la forma de producir y utilizar errores a título voluntario.



Al igual que cualquier otra tarea realizada por el hombre, la confección de programas para ordenador es también una actividad propensa a acoger errores. En tal caso, el ordenador manifestará un funcionamiento distinto al esperado.

Depuración de programas

En determinadas circunstancias el mal funcionamiento de un programa se debe a errores de codificación. Puede darse el caso de que la ejecución no pase nunca por una rutina determinada.



Es necesario revisar los programas para evitar errores de codificación. Por ejemplo, un salto a una línea que no existe.

En el otro extremo puede ocurrir que no se salga nunca de un bucle. En ambos casos, el problema se debe a errores en la desviación del flujo de ejecución.

En la etapa de creación de un programa es conveniente seguir una estructura definida. Delimitar los bloques de subrutinas y no mezclar las distintas partes del programa, proporcionará una mayor claridad al listado. Ha de evitarse la excesiva proliferación de instrucciones del tipo GOTO. Un abuso de estas instrucciones hace que un programa se vuelva difícilmente inteligible para cualquier persona, incluido el propio autor.

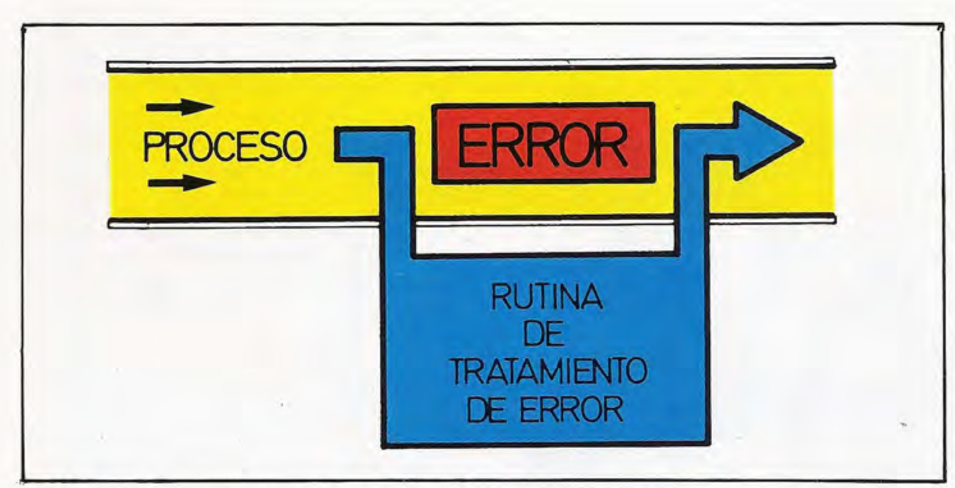
La forma más elemental de comprobar si un programa funciona, reside en el «seguimiento» de su ejecución «paso a paso» o «instrucción a instrucción». Este es el mejor procedimiento cuando se trata de comprobar el correcto funcionamiento de una rutina o de un programa corto.

La referida técnica consiste en actuar como lo haría el ordenador; esto es: ir leyendo las sucesivas intrucciones y ejecutarlas sobre el papel, una a una. Para ello es conveniente realizar una tabla de variables en la que se irán reflejando los sucesivos valores que adoptan. Cabe hablar de este método, pues, como de una simulación «a mano».

Realmente, se trata de una técnica muy adecuada; y no sólo para comprobar el buen funcionamiento de la rutina, sino también para comprender su modo de trabajo. El gran problema, no obstante, es que sólo es aplicable con plena eficacia cuando la rutina a estudiar es pequeña.

A título de ejemplo, veamos a continuación cómo se realizaría el seguimiento de una rutina corta, tal como la que sigue:

```
10 FOR I=1 TO 3
10 FOR J=1 TO 3
30 LET A(I,J)=I+J
40 NEXT J
50 NEXT I
█
```



Para subsanar de forma rápida y automática los previsibles errores que puedan producirse, es conveniente proceder a su tratamiento por medio de rutinas al efecto.

Para evaluarla aplicando la técnica descrita, se construye una tabla con las variables utilizadas. En este caso, se tienen las dos variables de los bucles y las nueve de la matriz. Se supone que la matriz está inicializada con ceros.

A medida que se va pasando por la línea 30, se van asignando valores a los elementos de A. En definitiva, la referida tabla de seguimiento de variables adoptará el aspecto que se indica en el cuadro adjunto.

En cada ejecución del bucle interno se asigna un valor a uno de los elementos de la matriz. Los restantes elementos no se alteran y siguen conservando los valores que tenían en la pasada anterior.

Otro procedimiento de uso también generalizado, consiste en introducir señalizadores en el programa que permitan saber por dónde está avanzando la ejecución del programa; ello permitirá saber si determinada rutina se ejecuta o no. Como señalizadores pueden emplearse varias de las posibilidades del aparato. Desde señales sonoras hasta mensajes que se han de visualizar en pantalla. Entre estos últimos cabe destacar como especialmente interesantes los consistentes en visualizar en pantalla los valores de determinadas variables.

Acto seguido se muestra un ejemplo de aplicación de esta técnica. Se trata de un programa que calcula los números primos hasta un cierto valor N definido por el usuario.

```
10 INPUT N
20 FOR I=1 TO N
30 GOSUB 1000
40 NEXT I
50 END
1000 PRINT "P"
1010 FOR J=2 TO I/2
1020 IF INT(I/J)=I/J THEN GOTO 1050
1030 NEXT J
1040 PRINT I
1050 RETURN
```

La instrucción PRINT "P" de la línea 1000 se ha introducido con el único objeto de comprobar si se han realizado tantas llamadas a la rutina como se esperaba a priori. Por cada P que aparezca en la pantalla sabremos que se ha efectuado una llamada a la subrutina. De este modo, se puede controlar si el flujo de ejecución del programa es el adecuado.

Otra forma útil de determinar el correcto funcionamiento del programa consiste en detener el proceso en el momento adecuado y examinar los valores de las variables en ese preciso instante. Téngase en cuenta que, por lo general, será posible continuar la ejecución del programa mediante el empleo del comando CONT.

También puede resultar de utilidad realizar ejecuciones parciales del programa dando unos valores determinados a ciertas variables. Al respecto, es conveniente recordar que una orden GOTO dirigida a un número de línea no borrará los valores previos de las variables, al contrario de lo que sucede al ejecutar un comando RUN.

Los comandos TRON/TROFF

Por último, dentro de la depuración de programas es interesante saber que muchos aparatos incorporan dos instrucciones muy útiles. Se trata de los comandos TRACE ON y TRACE OFF. El uso de estos comandos permite conocer en todo momento las líneas de programa que se están ejecutando. Su modo de funcionamiento es el siguiente: TRACE ON o TRON (dependiendo del equipo) activa un mecanismo por el que se visualizan en la pantalla los números correspondientes a las líneas que se van ejecutando. Por su parte, TRACE OFF o TROFF desactiva dicho mecanismo.

Las herramientas TRON/TROFF resultan muy útiles, especialmente en programas relativamente complicados en los que realizar el seguimiento «manual» resulta una ardua tarea. Da buenos resultados cuando el programa no funciona por quedarse estancado en un

TRON

Pone en marcha el mecanismo de trazado de la ejecución.

Formato: (número de línea) TRON

Ejemplos: 10 TRON
TRON

bucle sin salida. En tal caso, únicamente es necesario dejar que el programa se ejecute e ir tomando nota de los números de línea que aparecen. Hay que observar si éstos se repiten tras un determinado ciclo. Si ese ciclo es tal que impide la ejecución satisfactoria del programa, será ahí (en esos números de línea) donde se halla el error.

Tratamiento de errores

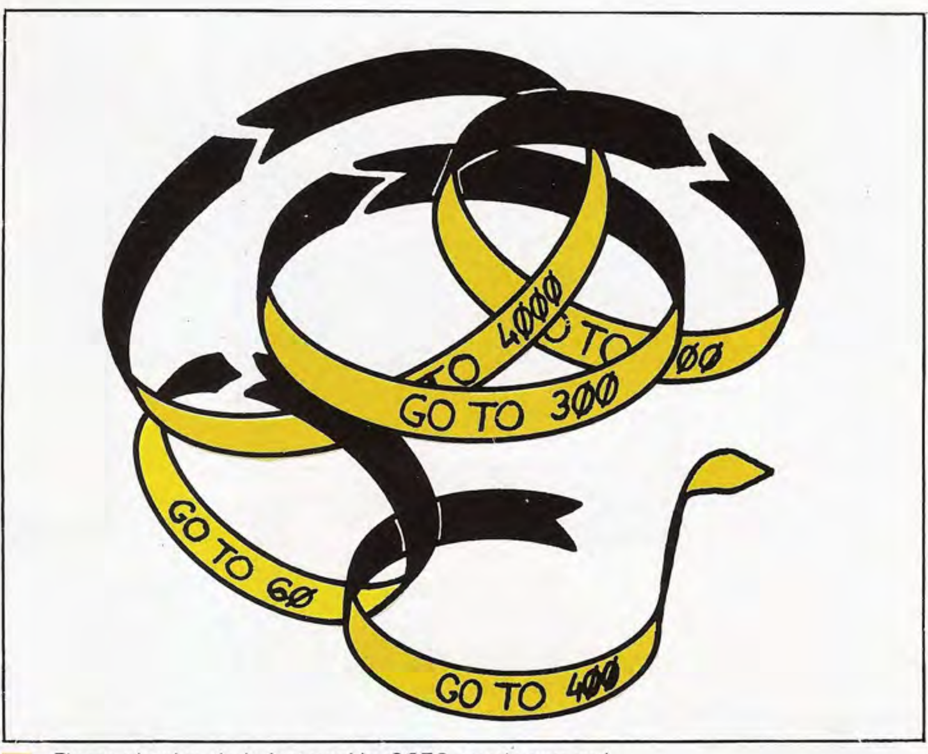
En condiciones normales, cuando se produce un error, la máquina informa de ello mediante el correspondiente mensaje de error y, por supuesto, deteniendo la ejecución del programa.

Esta forma de actuar suele ser la más conveniente en la mayor parte de las ocasiones. Sin embargo, ello no es siempre así. De hecho, este modo de proceder es molesto y desaconsejable en muchas ocasiones. Por ejemplo, se producirá un error al introducir un carácter alfabético tras la ejecución de un comando INPUT correspondiente a una variable numérica. Este error producirá la detención del programa, siendo necesario reiniciarlo si se desea continuar.

Existe una primera solución al problema. Esta consiste en realizar la introducción del dato como variable de cadena para, más tarde, operar la necesaria conversión. Existe otra solución, que es precisamente la que se pretende estudiar en este capítulo; solución que tiene como base el tratamiento del error por parte del programa. Para llevarlo a cabo es necesario indicar al ordenador que, en caso de producirse dicho error, se ejecute una determianda rutina. Rutina ésta que ha de aportar las instrucciones necesarias para el correcto tratamiento del error detectado. Esa rutina puede muy bien ser codificada en BASIC por el propio usuario. El comando necesario para indicar esta situación al ordenador es ON ERROR GOTO, cuyo formato es el que se muestra a continuación:

(Número de línea) ON ERROR GOTO <número de línea>

Donde <número de línea> corresponde al número de línea donde empieza la rutina que se ha de ejecutar en el caso de que se produzca el error en cuestión.



El uso abusivo de la instrucción GOTO puede convertir el seguimiento del programa en una auténtica odisea.

A partir de este momento es necesario conocer algunos detalles que intervienen en la rutina de tratamiento de error. Para empezar, existen dos variables: ERR y ERL que contienen información acerca del error que se ha producido. La primera de ellas es ERR (ERror Report) y contiene el código que identifica el error que ha tenido lugar. Por su parte ERL (ERror Line) contiene un número que corresponde al número de línea en el que se ha detectado dicho error. Con esta información ya es posible conocer unos parámetros bastante interesantes y que pueden ser empleados en la rutina.

Los códigos de error vienen asignados por el propio ordenador con el que se trabaje; aunque bien es cierto que también existen otros códigos no asignados que pueden ser empleados para definir errores de otra índole.

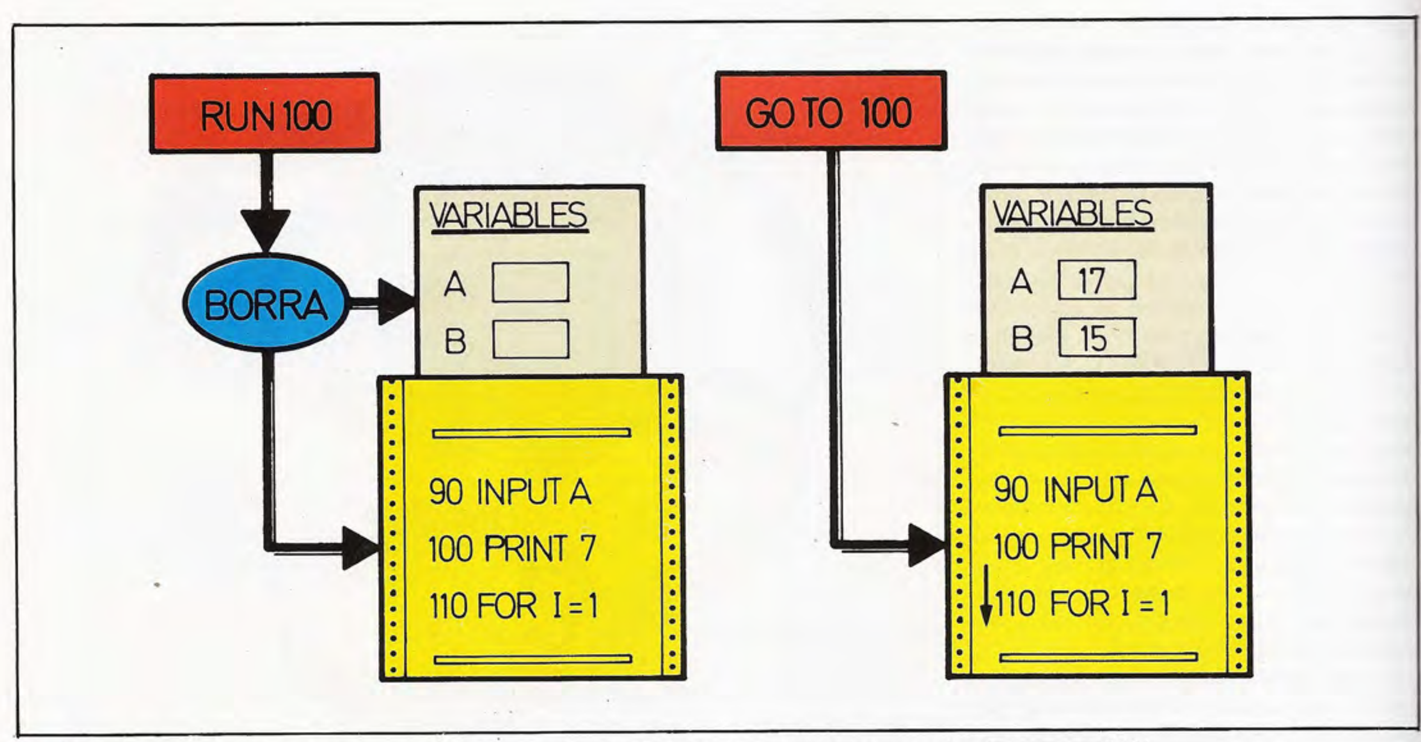
En cuanto al repetorio de errores que aporta el aparato y a sus códigos correspondientes, lo más efectivo es consultar el manual del propio equipo, ya que la equivalencia de esos códigos es específica de cada ordenador.

TROFF

Desactiva la ejecución paso a paso o trazado del programa.

Formato: (número de línea) TROFF

Ejemplos:
```
TROFF
70 TROFF
```



Después de interrumpir la ejecución de un programa, ésta puede reiniciarse por medio de la orden RUN o la GOTO. La diferencia entre ambas radica en el hecho de que RUN borrará todas las variables del programa, mientras que GOTO mantendrá los valores de las mismas.

Las rutinas de tratamiento de error han de incluir, al final, una instrucción que continúe la ejecución del programa. Estas rutinas actúan de forma muy parecida a las subrutinas invocadas con el comando GOSUB. Al igual que aquellas, necesitan de una instrucción que, como RETURN, devuelva el control a la zona principal del programa. Esta instrucción es RESUME y su formato es el siguiente:

(Número de línea) RESUME <opción>

En el que la zona de <opción> hace referencia a una de las posibles alternativas que a continuación se comentan.

- RESUME sin argumento o RESUME 0

Una vez procesada la rutina de tratamiento, si el ordenador se encuentra con esta instrucción continuará la ejecución del programa principal en la misma línea en la que se produjo el error.

- RESUME NEXT

Esta opción hace que, tras el tratamiento del error, la ejecución del programa continúe en la línea siguiente a aquella en la que se produjo dicho error.

- RESUME <número de línea>

Cuando se incluye este argumento, el programa continúa ejecutándose en la línea correspondiente al número que se especifica.

De esta forma, después de tratar el error se puede volver al programa principal, prosiguiendo su ejecución desde el punto deseado.

Cuando se emplea una de estas instrucciones sin que previamente se haya producido un error, se producirá a su vez otro error: RESUME ERROR. Por lo tanto, es preciso separar la línea que contenga este comando del resto del programa. Ese error es similar al que se produce al intentar ejecutar un comando RETURN sin haber efectuado previamente una llamada a subrutina (GOSUB).

Y no sólo se pueden tratar los errores contemplados por el aparato. También el propio usuario puede crear errores que atiendan situaciones no previstas por el fabricante. Esto último se consi-

ON ERROR GOTO

Hace que, al producirse un error, se ejecute la rutina situada a partir de la línea que se indica.

Formato: ON ERROR GOTO <número de línea>

Ejemplos:
```
10 ON ERROR GOTO 100
50 ON ERROR GOTO 2500
```

gue utilizando los códigos de error que no estén asignados previamente, o tratando de forma distinta los errores menos habituales.

ERROR

Produce el error identificado por el número de código indicado en su argumento.

Formato: (número de línea) ERROR <código de error>

Ejemplos:
```
20 ERROR 80
70 ERROR 2
```

Errores «a voluntad» del usuario

Para producir un error se puede emplear el comando ERROR, cuyo formato es el que se indica.

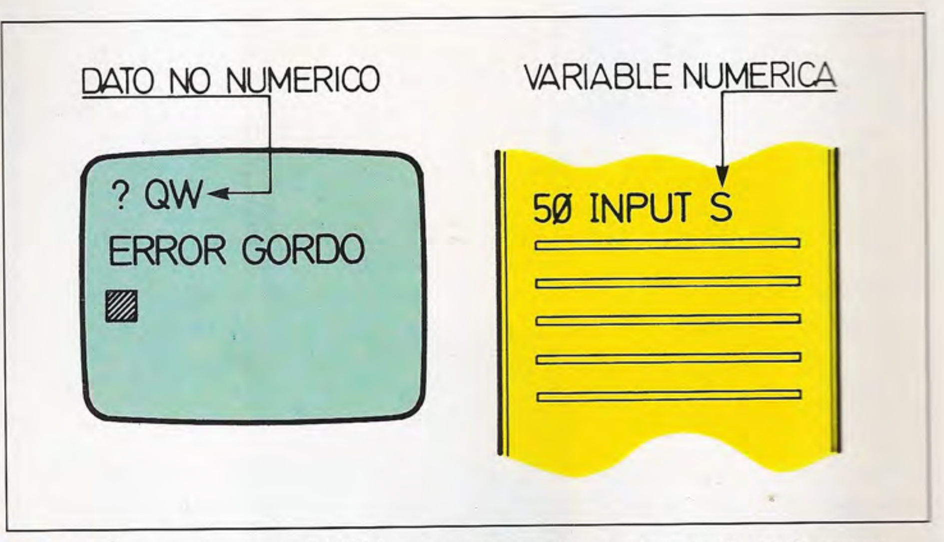
(Número de línea) ERROR <código de error>

Donde <código de error> es un dato numérico correspondiente al código que identifica al error deseado.

Con este comando se puede generar cualquiera de los errores contemplados en el aparato, así como otros que el usuario desee emplear. Estos últimos han de ser los correspondientes a los códigos que se encuentran sin asignar.

A continuación, se incluye un ejemplo relativo al uso de las instrucciones para el tratamiento de errores. En este caso se aplica al conocido juego de adivinar un número.

```
10 REM ADIVINA UN NUMERO
20 ON ERROR GOTO 80
30 LET A=I+INT(100*RND)
40 INPUT "NUMERO";B
50 IF A>B THEN ERROR 80 ELSE IF A<B THEN ERROR 81
60 PRINT "ACERTASTE"
70 END
80 IF ERR=81 THEN PRINT "DEMASIADO ALTO" ELSE
   PRINT "DEMASIADO BAJO"
90 RESUME 40
100 END
```





El propio intérprete de BASIC hace uso de un sistema de tratamiento de errores. Esto se pone de manifiesto, por ejemplo, al intentar asignar una cadena de caracteres a una variable numérica.

En la línea 20 se comunica al ordenador que, en el caso de producirse un error, debe desviar la ejecución a una rutina de tratamiento situada a partir de la línea 80. A continuación, en la línea 30 se genera el número que el usuario debe acertar. Este número estará comprendido entre uno y cien.

En la línea 40 el aparato pide al operador que introduzca el número que constituye su intento; mientras que en la línea 50 se comprueba si el número introducido coincide con el que calculó la máquina. En ese preciso momento se puede generar un error, que será el 80 si el número introducido es mayor que el calculado por la máquina, o el 81 si es menor.

Si se produce uno de ambos errores, el ordenador efectuará un alto en la ejecución al número de línea indicado en la instrucción ON ERROR GOTO. El comando situado en la línea 60 es el que se ejecutará cuando se acierte el número; su misión es presentar en la pantalla un mensaje de felicitación. Por último, la instrucción de la línea 70 es la que finaliza la rutina principal.

Entre las líneas 80 y 100 está situada la rutina de tratamiento de errores. En ella se encuentra la línea 80, que genera un mensaje si el error que se está tratando es el 80, y otro si corresponde al 81. Por último, la línea 90 es la que pone fin a la rutina y produce el retorno a la zona principal. El retorno se efectúa, concretamente, a la línea 40.

Si el ordenador utilizado no dispone de la partícula ELSE dentro del comando IF, se puede sustituir la línea 50 por las dos siguientes:

```
50 IF A>B THEN ERROR 80
55 IF A<B THEN ERROR 81
```

De la misma forma, la línea 80 se verá sustituida por las que se indican a continuación:

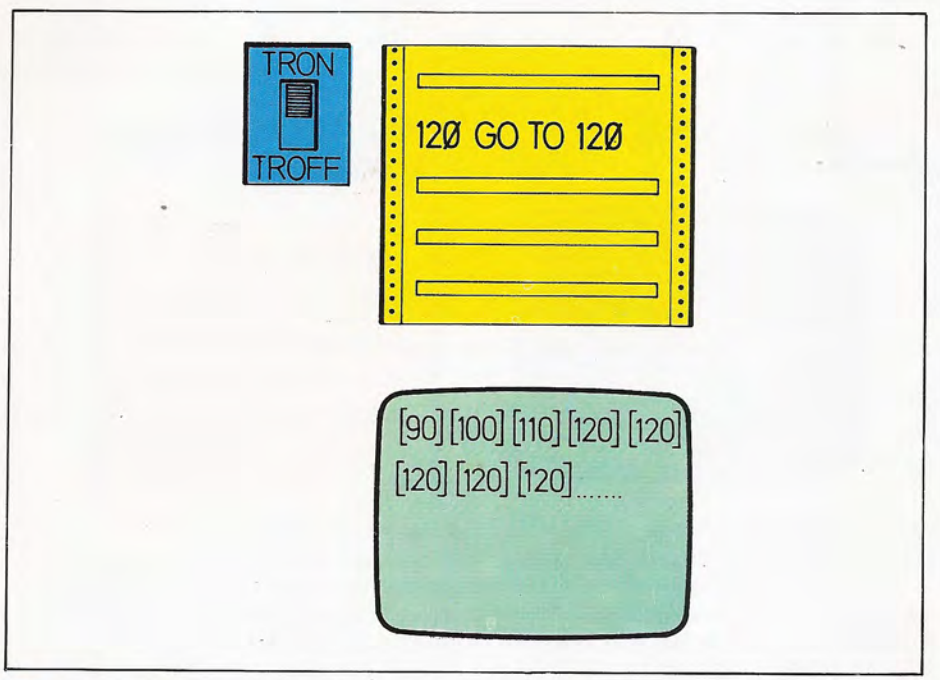
```
80 IF ERR=81 THEN PRINT "DEMASIADO ALTO"
85 IF ERR=80 THEN PRINT "DEMASIADO BAJO"
```

RESUME

Reanuda la ejecución de la zona principal del programa en el punto indicado en su argumento.

Formato: (número de línea) RESUME <opción>

Ejemplos: 20 RESUME
70 RESUME 50



Las instrucciones TRON y TROFF activan y desactivan el trazado del programa o, lo que es lo mismo, su ejecución paso a paso. Esta posibilidad resulta muy útil para seguir el flujo de ejecución a través de las diversas zonas del programa. Ello permitirá detectar fácilmente la existencia de bucles infinitos.

I	J	A(1,1)	A(1,2)	A(1,3)	A(2,1)	A(2,2)	A(2,3)	A(3,1)	A(3,2)	A(3,3)
1	1	2	0	0	0	0	0	0	0	0
1	2	2	3	0	0	0	0	0	0	0
1	3	2	3	4	0	0	0	0	0	0
2	1	2	3	4	3	0	0	0	0	0
2	2	2	3	4	3	4	0	0	0	0
2	3	2	3	4	3	4	5	0	0	0
3	1	2	3	4	3	4	5	4	0	0
3	2	2	3	4	3	4	5	4	5	0
3	3	2	3	4	3	4	5	4	5	6

Tabla de evolución de variables asociada al seguimiento de la rutina BASIC incluida en el texto.

El tratamiento de errores en la práctica

Como ya se ha indicado, los errores de usuario pueden ser empleados para producir y tratar errores no contemplados en el repertorio propio del aparato.

Por ejemplo, si como se comentaba más arriba se desea que no se detenga la ejecución del programa cuando se produzca un error, se puede emplear la técnica que revela el siguiente ejemplo:

```
 90 ON ERROR GOTO 100
100 INPUT A
110 PRINT 100/A
```

En este caso, al producirse un error se saltará a la línea 100. El error puede originarse al intentar asignar el valor cero a la variable numérica A. Al tener lugar ese error se salta a la línea 100 volviéndose a recoger un nuevo valor del teclado. Con ello se consigue la rei-



La instrucción RESUME indica al ordenador en qué punto del programa principal ha de continuar la ejecución tras procesar la rutina para el tratamiento del error.

TABLA DE CONVERSION					
Ordenador	TRAZADO	TRATAMIENTO DE ERRORES			
	TRON/TROFF	ON ERROR GOTO n	RESUME	ERROR n	ERR y ERL
AMSTRAD	TRON/TROFF	ON ERROR GOTO n	RESUME	ERROR n	ERR y ERL
APPLE II (APPLESOFT)	TRACE/NO TRACE	ON ERR GOTO	RESUME	—	—
APRICOT (M-BASIC)	TRON/TROFF	ON ERROR GOTO n	RESUME	ERROR n	ERR y ERL
ATARI	—	TRAP	—	—	—
CBM 64	—	—	—	—	—
DRAGON	TRON/TROFF	—	—	—	—
EQUIPOS MSX	TRON/TROFF	ON ERROR GOTO n	RESUME	ERROR n	ERR y ERL
HP-150	TRON/TROFF	ON ERROR GOTO n	RESUME	ERROR n	ERR y ERL
IBM PC	TRON/TROFF	ON ERROR GOTO n	RESUME	ERROR n	ERR y ERL
MPF	TRACE/NO TRACE	ONERR GOTO	RESUME	—	—
NCR DM-V (MS-BASIC)	TRON/TROFF	ON ERROR GOTO n	RESUME	ERROR n	ERR y ERL
NEW BRAIN	—	ON ERROR GOTO n	RESUME	ERROR n	ERRNO/ERRLIN
ORIC	TRON/TROFF	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	—	ON ERROR GOTO n	RESUME	—	ERR y ERL
SINCLAIR QL	—	—	—	—	—
SPECTRAVIDEO	TRON/TROFF	ON ERROR GOTO n	RESUME	ERROR n	ERR y ERL
ZX-SPECTRUM	—	—	—	—	—

terada ejecución del comando INPUT. El programa sólo continuará si el dato introducido no conduce a error.

Realmente, en el ejemplo propuesto no existe un tratamiento del error propiamente dicho. Para efectuar dicho tratamiento habría que utilizar una rutina que podría ser semejante a la incluida en las líneas 1000 y 1010 del siguiente ejemplo:

```
90 ON ERROR GOTO 1000
100 INPUT A
110 PRINT 100/A
.................................
.................................
.................................
900 END
1000 PRINT "DATO NO VALIDO, INTENTELO DE NUEVO"
1010 RESUME 100
```

Como se observa, el tratamiento del error consiste en mostrar un mensaje y repetir la introducción del dato. Hay que poner cuidado en la elección del punto de retorno. Si la línea 1010 incluyera un comando RESUME sin argumento, se reanudaría la ejecución en la línea 110. Ello provocaría un retorno a la línea que da origen al error, sin subsanar el mismo.

En el caso de que se pueda conocer a priori el tipo de error originable, éste puede ser corregido en la rutina de tratamiento. En el ejemplo se sabe que el error se producirá cuando A valga cero; por lo tanto, la rutina de error puede utilizarse para cambiar el valor de dicha variable. Esto es precisamente lo que se hace en el siguiente ejemplo:

```
90 ON ERROR GOTO 1000
100 INPUT A
110 PRINT 100/A
.................................
.................................
.................................
900 END
1000 LET A=1
1010 RESUME
```

Ahora, de producirse el error, será el mismo programa quien lo solucione, asignando un valor no nulo (en este caso 1) a la variable A.

Si se prevén distintos tipos de errores, se pueden separar sus rutinas de tratamiento de varias formas.

Si los diferentes errores surgen en zonas separadas del programa, lo más adecuado es intercalar comandos ON ERROR GOTO con distintos argumentos. Así, al principio de una determinada zona de instrucciones, se colocará una orden del tipo ON ERROR GOTO que salte a la rutina que atienda el error propio de esa parte del programa. Ello supone añadir tantas rutinas de tratamiento como errores se prevean.

En el caso de que el error pueda surgir en cualquier parte del programa y pueda ser de distinto tipo, lo anterior será inviable. Esto ocurrirá si en una misma línea se pueden producir errores distintos. La solución en tal caso es utilizar una rutina de tratamiento que "distinga" el tipo de error. Dicha distinción se puede realizar muy fácilmente utilizando la variable ERR para bifurcar la ejecución.

También es posible identificar de una manera más detallada el error. Esto último se consigue con el empleo conjunto de ERR y ERL.

De esta forma se puede realizar un tratamiento individualizado de cada error que se produzca.

Códigos detectores de error

Paridad par y paridad impar

En la transmisión de información de un emisor a un receptor pueden producirse errores. Esto ocurre en todo tipo de comunicaciones. En una conversación entre dos personas es posible que el ruido ambiente haga difícil el entendimiento entre éstas. Al comunicarse utilizando otro medio de transmisión también se dan interferencias que alteran la información transmitida. El ejemplo más palpable es el de las conferencias telefónicas.
En el interior del ordenador se produce un continuo trasiego de información. Esta se va desplazando del teclado a la unidad central, de la unidad central a la memoria, de la unidad central a la pantalla, de la memoria a la unidad central, etc. En este maremagnum de tráfico no es extraño que se produzcan alteraciones en la información.
Como es sabido, la información es tratada por el ordenador en forma de impulsos eléctricos. Dichos impulsos se transmiten a través de conductores y cuanto mayor es la longitud del conductor, mayor resistencia opone éste al paso de la corriente. Esto hará que algunos impulsos queden anulados, produciéndose errores en la transmisión.
Para paliar de algún modo esos errores de transmisión se hace uso de los sistemas de detección de error. Para ello se recurre al envío de información redundante que permite la confrontación de los datos. Si una información no coincide con la otra es que se ha producido un error. Los diferentes métodos detectan el error producido en la transmisión de un byte. La información adicional se codifica en algunos bits que se añaden a los ocho de cada byte.
El sistema más sencillo y más empleado es el método de la paridad. En él se comprueba si el número de unos transmitidos es par o impar. La paridad par consiste en añadir a la codificación de cada uno de los caracteres un bit; bit que se elige de forma que el número total de unos sea par. Con este convenio es posible detectar si se ha producido un error en uno de los bits. Veamos cómo.
Suponga que se desea transmitir la siguiente información:

01010101
11100000
11111110

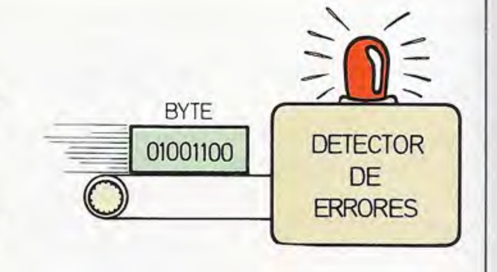
A estos bytes es preciso añadirles un noveno bit; bit que hara que el número de unos sea par. Los nuevos datos serían los siguientes:

01010101 0
11100000 1
11111110 1

Tal es la información que se envía. En el extremo receptor se verifica la paridad de los datos; si el número de unos de alguno de los bytes es impar es que existe un error; a su vez, si no hay error se elimina ese noveno bit.



Los errores pueden convertir en ininteligible a una transmisión. Para evitar esta situación se recurre al empleo de los bits de paridad, cuya misión es la de permitir la detección de posibles errores en la información transferida.



Con los datos anteriores se podría recibir la secuencia que se muestra a continuación.

00010101 0
11100000 1
11111110 1

Al contar el número de unos del primer byte de datos se ve que es impar. Esto permite asegurar que se ha producido un error en la transmisión de dicho dato. Este método de detección sólo es seguro cuando se produce un único error por byte. Si se producen dos errores (o un número par de errores) éstos no serían detectados. Veamos lo que ocurre cuando se introduce un segundo error en el dato anterior.

10010101 0

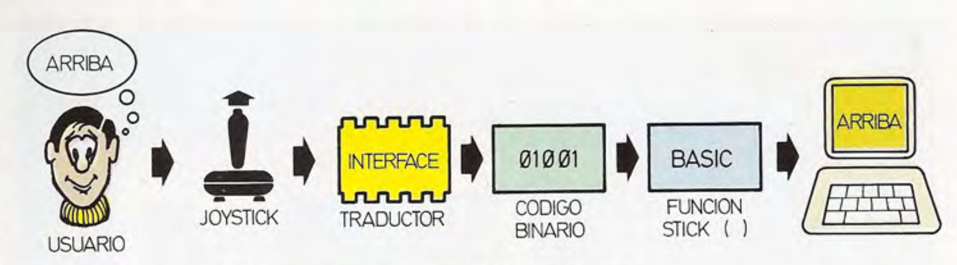
El dato no coincide con el enviado por el emisor. Sin embargo, el número de unos es par. Esto indica al receptor que no existe error, cuando sabemos que ello no es cierto.
Afortunadamente, en los sistemas modernos es cada vez más difícil que se produzca un error de transmisión. Si la probabilidad de que coincidan dos en el mismo byte es mucho más remota. Por ello, este sistema es suficientemente eficaz.
El método de paridad impar es semejante al comentado. La única diferencia estriba en la elección del noveno bit. En el caso de paridad impar, el mencionado bit ha de hacer que el número de unos de cada byte de datos sea impar.

Control de periféricos

Joysticks y *paddles* en acción



ara lograr una mayor verosimilitud y realismo en la ejecución de programas de acción, el ordenador necesita recibir informes y sensaciones externas que le indiquen qué es lo que está ocurriendo a su alrededor. Estas influencias del medio exterior serán captadas por el ordenador en forma de señales eléctricas, a través de dispositivos adaptadores denominados *interfaces.* En resumidas cuentas, los dispositivos de *interface* no son más que fieles traductores de esas señales eléctricas, procedentes de algún periférico, a un formato comprensible por el ordenador.



Para que una orden que parte de la mente del usuario llegue a ser «comprendida» por el ordenador, es necesaria la intervención de una serie de dispositivos intermedios.

Para comprender mejor el proceso de captación de información del exterior, se puede pensar en la secuencia que se seguirá para conseguir que una determinada orden llegue a ser «comprendida» por la máquina. En primer lugar, el usuario decidirá qué orden desea transferir a la máquina. Acto seguido emprenderá las acciones necesarias, actuando sobre algún dispositivo periférico externo al ordenador que transforme sus pensamientos en señales eléctricas. A continuación, el periférico pasará esta información a un *interface* que se encargará de convertir las señales eléctricas en señales digitales codificadas en binario (unos y ceros), que son las únicas que puede comprender directamente el ordenador. Una vez que el ordenador ha recibido ese código binario, sólo le queda interpretarlo en la forma adecuada, para poder ejecutar la orden que quería transmitirle el usuario. Pero, ¿quién se encarga de interpretar los códigos binarios recibidos? Como es fácil suponer, será el propio ordenador instruido por el programa que se esté ejecutando en ese momento; programa que, normalmente, estará codificado en BASIC.

Así pues, es obvio que el BASIC dispondrá de diversas funciones y comandos para poder controlar los distintos periféricos que se pueden acoplar a un ordenador; el estudio de estas funciones y comandos constituye, precisamente, el objetivo de este capítulo.



Moviendo la palanca de mando, el usuario puede comunicar al ordenador con celeridad órdenes que, normalmente, afectarán al desplazamiento de un objeto sobre la pantalla.

La palanca de mando o *joystick*

Entre la gran gama de periféricos de «acción» que se pueden acoplar a un ordenador, el más popular es sin duda alguna la palanca de mando o *joystick.* Este consta de una base sobre la que se eleva una palanca móvil, similar a la palanca de mando de los aviones de combate. Su utilidad a la hora de controlar la acción asociada a los programas de juegos es indudable... Pero, ¿cómo funciona un *joystick?*

Su cometido más frecuente es indicar la dirección y sentido hacia el que se quiere desplazar por la pantalla una nave espacial o cualquier otro artefacto similar. Mediante el movimiento de la palanca en cualquiera de las cuatro direcciones cardinales (norte, sur, este y oeste), o sus direcciones intermedias (noreste y noroeste, sureste y suroeste), se consiguen definir ocho posibles orientaciones para el movimiento.

Para detectar la dirección en que ha sido movida la palanca, se dispone de cuatro interruptores situados en la base, uno para cada «punto cardinal». Así, al mover la palanca en uno de estos sentidos se cerrará el correspondiente interruptor, generando una señal eléctrica que es codificada por el *interface;* este circuito adaptador proporciona así un



El joystick o palanca de control es el periférico lúdico por excelencia; su presencia es muy frecuente junto a los ordenadores domésticos.

código binario diferente para cada sentido de desplazamiento. El código binario puede ser «leído» desde el BASIC con la ayuda de la función STICK ( ), que retornará un valor diferente para cada una de las ocho posibles direcciones. Como sucede con todas las funciones BASIC, dicho valor debe ser asignado a alguna variable previamente definida para que esté disponible para su uso posterior.

El formato más general de una instrucción encargada de «leer» el estado en cada momento de la palanca de mando, es el que sigue:

<Núm. de línea> X=STICK (<exp.>)

El formato incluye a la variable X (que puede ser otra cualquiera, con otro nombre, siempre que se trate de una variable numérica), en la que se almacena el valor proporcionado por la función STICK; dicha variable será utilizada en el resto del programa.

La expresión <exp.> tiene la misión de diferenciar el *joystick* cuyo estado se quiere examinar, puesto que generalmente será posible conectar con simultaneidad dos palancas de mando en sendos conectores de entrada. Bien es cierto que en algunos casos será posible conectar hasta cuatro palancas de mando o más. En definitiva, el *joystick* conectado a la toma 1 se leerá normalmente con: X=STICK(1); mientras que la palanca conectada al acceso 2 se leerá con: X=STICK(2); y así sucesivamente. La expresión utilizada puede ser cualquier constante, variable o expresión matemática, que se evaluará al ejecutar la instrucción para obtener así un número entero, como se observa en el siguiente ejemplo:

```
X=STICK(Y—2)
```

En él, si el valor actual de Y es 4, se leerá el *joystick* conectado al acceso o toma 2. Ello significa que se puede transferir el control a otra palanca con sólo variar el valor de Y. Generalmente, el rango de posibles valores del parámetro asociado a la función STICK está acotado entre ciertos límites, generando un mensaje de error (de llamada ilegal a una función) si se intenta una llamada con un valor fuera de rango. Por desgracia, el parámetro <exp.> que se debe aportar a la función STICK ( ), así como los valores entregados por dicha función, varían mucho de unos intérpretes BASIC a otros, por lo que será imprescindible consultar el manual de cada ordenador.

En algunos casos (en los equipos MSX, por ejemplo), es posible leer con la función STICK las teclas de movimiento del cursor, simulando así una palanca de mando que permitirá desplazar objetos por la pantalla. Como es fácil imaginar, se asignará a esta función particular un número de «port» o acceso (<exp.>) exclusivo para este cometido; por ejemplo: X=STICK(0).

¿Cómo se pueden diferenciar las diversas acciones a tomar según el código entregado por la función STICK ( ) en el contexto de un programa BASIC? Normalmente, esto se hace por medio de instrucciones de decisión del tipo IF/THEN/ELSE: según se cumpla o no una determinada condición, se podrá emprender consecuentemente una acción u otra.

Para poder dar los siguientes pasos habrá que conocer los códigos que genera la función STICK, para lo que será imprescindible consultar el manual de cada dialecto BASIC o bien acudir al siguiente programa de prueba:

```
10 X=STICK (1)
```

STRIG

Lectura del botón de disparo del *joystick* conectado al «port» especificado.

Formato: [<var.>=] STRIG (<exp.>)

Ejemplo: 20 Y=STRIG (2)

STICK

Lee del «port» o toma de entrada que se especifique, el código que define la posición actual del *joystick*.

Formato: [<var>=] STICK (<exp.>)

Ejemplo: 30 X=STICK (1)

El «joystick» es un periférico lúdico que permite la transmisión directa de órdenes a la máquina; órdenes destinadas normalmente a controlar el movimiento de figuras por la pantalla.

```
20 PRINT X
30 GOTO 10
```

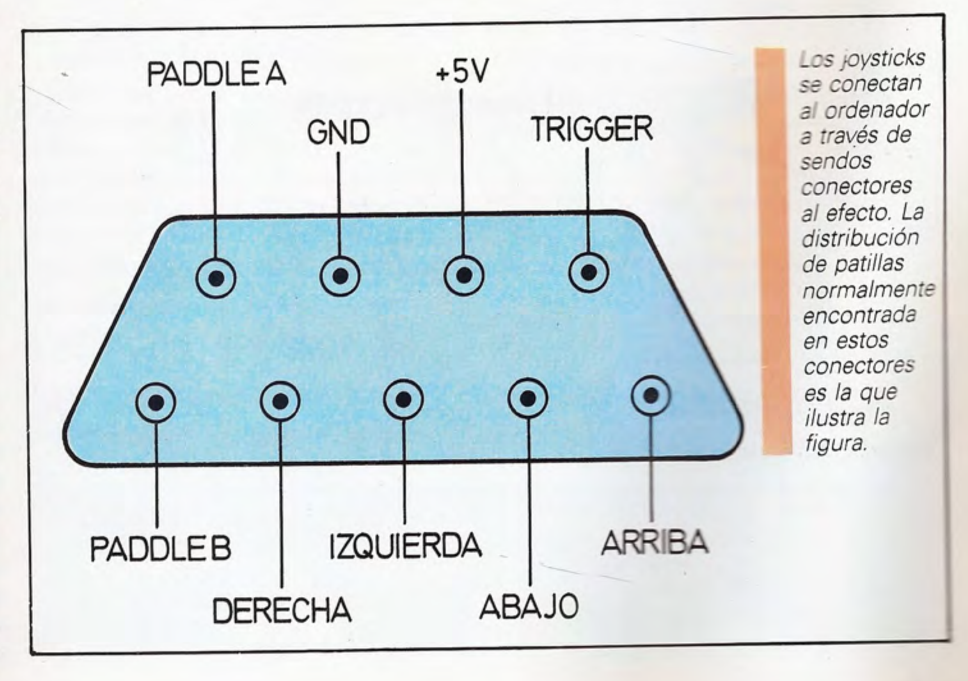
Al ejecutar este pequeño programa se imprimirá de forma continua en la pantalla el valor leído en cada instante por la función STICK(1), correspondiente al código de la posición en que se encuentre situado el *joystick* conectado a la toma 1. Para tener un punto de referencia, se supondrá que se dispone de un *joystick* imaginario que entrega los siguientes códigos para cada una de las posibles direcciones de la palanca de mando:

DIRECCION	CODIGO
REPOSO:	0
NORTE:	1
NORESTE:	2
ESTE:	3
SURESTE:	4
SUR:	5
SUROESTE:	6
OESTE:	7
NOROESTE:	8

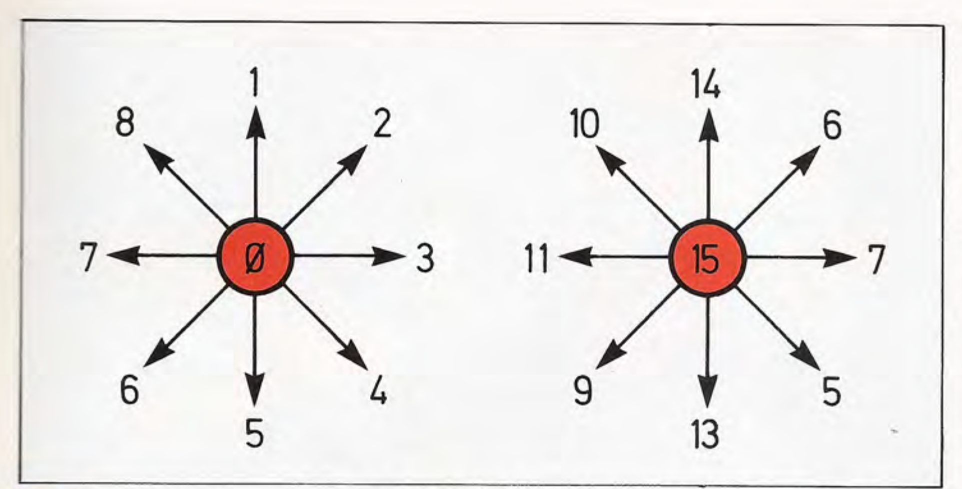
Bajo este supuesto, se pueden ya codificar las diferentes acciones a tomar

En el siguiente ejemplo cuyo listado (PROGRAMA 1) se reproduce junto al texto, se puede ver la utilización de sentencias IF/THEN junto con la función STICK ( ); el objetivo es, sencillamente, mover el carácter asterisco ("*") por toda la pantalla bajo las órdenes del *joystick*.

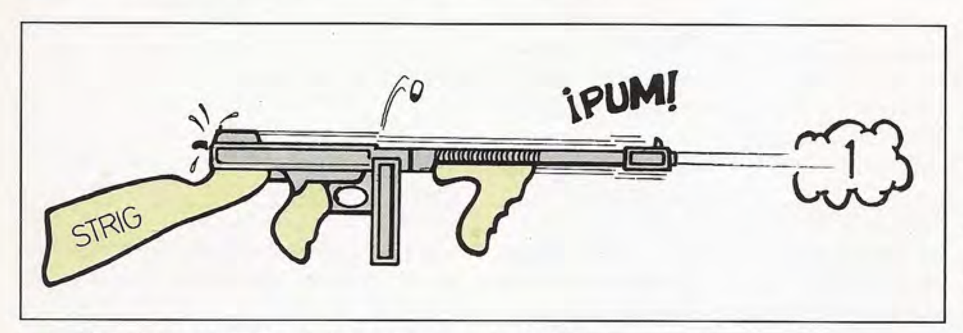
El funcionamiento de este programa debe resultar de muy fácil asimilación a



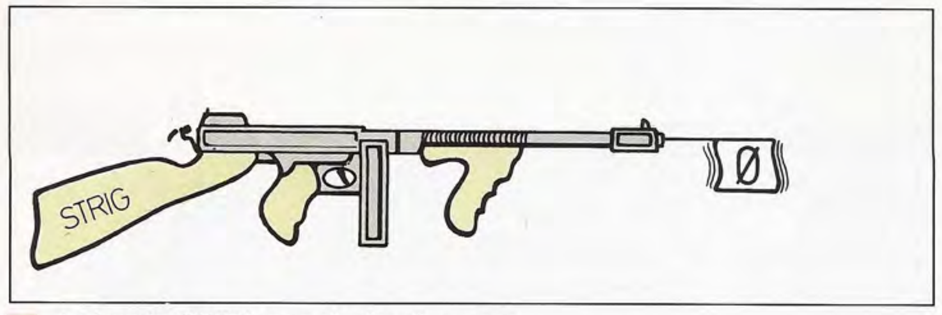
Los joysticks se conectan al ordenador a través de sendos conectores al efecto. La distribución de patillas normalmente encontrada en estos conectores es la que ilustra la figura.



Los códigos generados por la función STICK() varían mucho de un ordenador a otro. La figura muestra dos de las distribuciones más comunes; los números que aparecen en los extremos de las flechas corresponden a los códigos asociados a cada sentido de desplazamiento de la palanca de mando.



Al accionar el botón de disparo, la función STRIG() devuelve el valor «1».



La función STRIG() toma el valor «0» cuando no se está pulsando el botón de disparo.

ON STRIG GOSUB

Bifurca a una subrutina al producirse una interrupción provocada por un botón de disparo del *joystick*.

Formato: ON STRIG GOSUB <nl.> [,..., <nl.>]

Ejemplos: 10 ON STRIG GOSUB 1000
20 ON STRIG GOSUB 1000, 2000, 3000

estas alturas del curso, por lo que tan solo basta con precisar que está escrito para un ordenador cuya visualización en pantalla es de 24 filas por 40 columnas en modo de texto, y con el origen de coordenadas para el comando PRINT AT (X,Y) situado en el vértice superior izquierdo de la pantalla. Si su ordenador no satisface este supuesto, será necesario adaptar el programa a las especificaciones de cada caso concreto, lo que no tiene mayor dificultad. Así mismo, habrá que modificar en la mayoría de los casos los valores evaluados por las instrucciones IF/THEN de las líneas 70 a la 170.

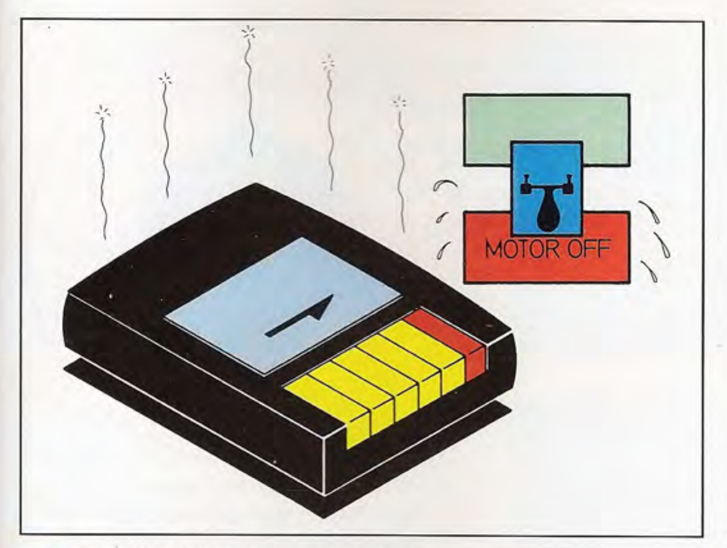
Joystick y botón de disparo

Por el momento no se ha mencionado aún otro de los elementos que incorpora toda palanca de mando y que es sumamente útil para muy diversas funciones: el botón de disparo. Se trata de un pulsador que puede estar situado en diferentes puntos del *joystick* —en lo alto de la palanca o en su parte anterior a modo de gatillo, o incluso en la base del *joystick*—, y al que por norma se le suele asignar el papel de «gatillo» disparador para lanzar mortíferos misiles contra los más extraños alienígenas. En todo caso, y como se verá más adelante, su cometido no se limita únicamente al mencionado.

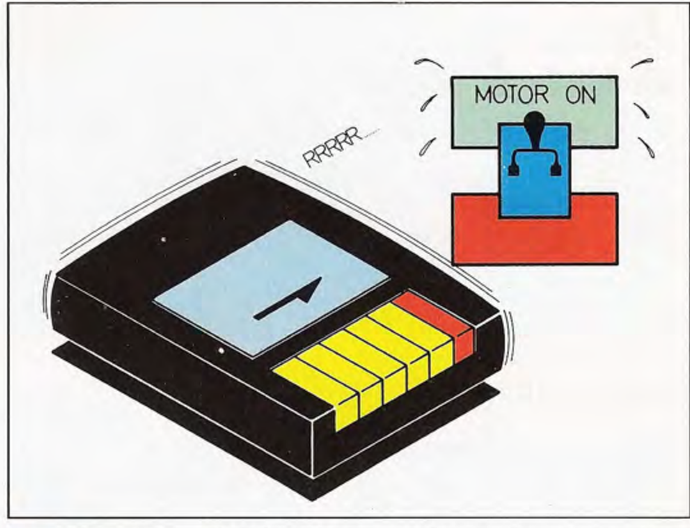
En su aspecto *hardware*, el botón de disparo es totalmente análogo a los contactos de la base de la palanca de mando, pero en este caso sólo existe un contacto, que según esté pulsado o no, permite diferenciar los estados activo e inactivo. Ello se refleja en una señal eléctrica que es tratada normalmente por el mismo *interface* del *joystick*. La función BASIC que «lee» el estado del botón de disparo es STRIG ( ), cuyo formato más general es el siguiente:

<Núm. de línea> X=STRIG (<exp.>)

Su funcionamiento, utilización y parámetros son totalmente análogos a los de la función STICK, salvo en lo relativo a que la función STRIG sólo proporciona dos posibles valores, según esté o no pulsado el botón de disparo. Estos dos valores suelen ser: «1» cuando está pulsado el botón, y «0» cuando no está siendo accionado, o viceversa. Al igual que ocurría con la función STICK, estos va-



MOTOR OFF apaga o desconecta el motor de arrastre del casete.



El comando MOTOR ON conecta el dispositivo de arranque del motor de arrastre del magnetófono a casete.

lores suelen variar bastante de unos intérpretes a otros, por lo que de nuevo se hace necesaria una consulta al manual BASIC de cada equipo concreto. Desde luego, también puede ejecutarse un sencillo programa, similar al realizado en el caso de la función STICK, para conocer el código entregado por STRIG; programa que puede servir también para comprobar el correcto funcionamiento del botón de disparo:

```
10 X=STRIG(1)
20 PRINT X
30 GOTO 10
```

Siguiendo con la plena analogía de esta función con respecto a la comentada anteriormente, se puede ver en el siguiente ejemplo como también el estado del botón de disparo se puede examinar mediante una instrucción IF/THEN, que permitirá emprender la acción deseada:

```
10 CLS
20 PRINT "TODAVIA NO SE HA PULSADO EL BOTON"
30 IF STRIG(1)=0 THEN GOTO 30
40 CLS
50 PRINT "YA SE HA PULSADO EL BOTON"
60 END
```

La función STRIG se puede equiparar por tanto a un «conmutador *software*», que permite realizar una acción específica, saltar a otra zona del programa, o ejecutar una subrutina determinada, cuando sea accionado el «conmutador». Pero estas acciones sólo se pueden realizar cuando la ejecución del programa llegue a una línea BASIC que tome una decisión basada en el examen de la función STRIG.

Control de interrupciones

Si quiere disponer de un «interruptor *software*», para realizar una determinada acción en el transcurso de la ejecución de un programa, se debe acudir a otra instrucción BASIC cuyo formato es el siguiente:

<Núm. de línea> ON STRIG GOSUB <núm.> [, <núm.>...]

Esta instrucción genera una interrupción del procesador en el preciso instante en que alguno de los botones de disparo es accionado, sin esperar a llegar a la línea que examina el resultado de la función STRIG. Con esto se consigue una respuesta inmediata, pasando el ordenador a ejecutar directamente la subrutina indicada. Como normalmente existen varias tomas de entrada a las que se pueden conectar los *joysticks*, se pueden situar detrás de la palabra GOSUB varios números de línea, separados por comas; se ejecutará la subrutina cuyo número de línea aparezca en el lugar correspondiente del número de orden de la toma a la que está conectado el botón de disparo accionado para interrumpir el programa. Así, en el siguiente ejemplo:

```
10 ON STRIG GOSUB 1000,2000,3000
```

se ejecutará la subrutina de la línea 1000 si se acciona el botón de disparo conectado a la toma 0; la subrutina de la línea 2000, si se acciona el botón de disparo de la toma 1, y la de la línea 3000 si se actúa sobre el de la toma 2. Cuando se llega a la sentencia RETURN de estas subrutinas, el control retorna

STRIG ON/OFF/STOP

Permite, prohíbe o desactiva momentáneamente la interrupción que provoque el botón de disparo especificado. Está relacionada con el comando ON STRIG GOSUB.

Formato: STRIG (<exp.>) [ON/OFF/STOP]

Ejemplos:
```
10 STRIG ON
20 IF I=Ø THEN STRIG OFF
50 STRIG STOP
```

PADDLE

Lectura del código generado por un *paddle* o «raqueta» cuyo número de «port» se especifica.

Formato: [<var.>=] PADDLE (<exp.>)

Ejemplos:
```
10 X=PADDLE (1)
20 IF PADDLE (2) > THEN GOTO 20
```

MOTOR ON/OFF

Controla el motor de un casete de audio, activando o desactivando un relé.

Formato: MOTOR [ON/OFF]

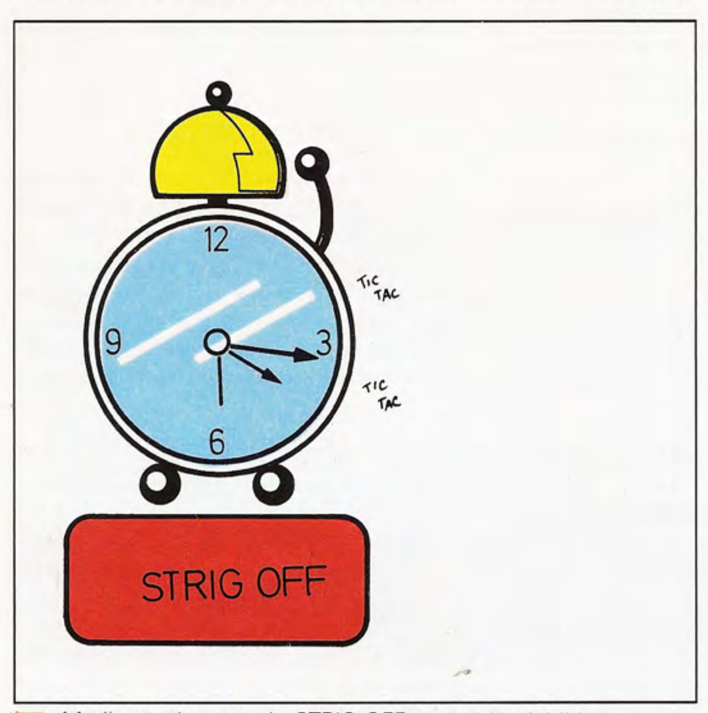
Ejemplos:
```
10 MOTOR ON
20 IF VAR=Ø THEN MOTOR OFF
```



El PADDLE o potenciómetro continuo genera un valor comprendido entre dos límites que es leído por medio de la función PADDLE.

al punto del programa en el que se produjo la interrupción, continuando la ejecución normalmente.

Pero este tema no acaba aquí, sino que también se pueden controlar estas interrupciones programadas por medio de la instrucción ON STRIG GOSUB. Después de haber ejecutado esta instrucción, se podrá hacer que sea efectiva la interrupción originada por la pulsación del botón de disparo, se podrán anular las interrupciones, o bien se podrán «congelar» temporalmente, produciéndose la interrupción una vez que sean nuevamente permitidas. Con ello se tendrá un control total del programa, de forma que en ciertos momentos en los que no convenga que se produzca una interrupción, éstas sean anuladas o bien «memorizadas» temporalmente, para «acusarlas» una vez finalizada la



Mediante el comando STRIG OFF se pueden inhibir o desactivar las interrupciones generadas por el botón de disparo a través de la sentencia ON STRIG GOSUB.



El comando STRIG ON permite que se produzcan interrupciones mediante la pulsación del botón de disparo del joystick.

TABLA DE CONVERSION						
Ordenador	STICK	STRIG	ON STRIG GOSUB	STRING ON/ OFF/STOP	PADDLE	MOTOR ON/OFF
	STICK(<n>)	STRIG(<n>)	ON STRIG GOSUB <nl.>[,...,<nl.>]	STRIG(<n>)ON/ OFF/STOP	PADDLE(<n>)	MOTOR ON/OFF
AMSTRAD	—	—	—	—	—	—
APPLE II (APPLESOFT)	—	—	—	—	PDL(<n>)	—
APRICOT (M-BASIC)	—	—	—	—	—	—
ATARI	STICK(<n>)	STRIG(<n>)	—	—	PADDLE(<n>)	—
CBM 64	—	—	—	—	—	—
DRAGON	JOYSTK(<n>)	—	—	—	—	MOTOR ON/OFF
EQUIPOS MSX	STICK(<n>)	STRIG(<n>)	ON STRIG GOSUB<nl.>[,...,<nl.>]	STRIG(<n>)ON/OFF/STOP	PDL<(<n>)	MOTOR ON/OFF
HP-150	—	—	—	—	—	—
IBN PC	STICK(<n>)	STRIG(<n>)	ON STRIG(<n>) GOSUB<nl.>	STRIG(<n>)ON/OFF/STOP	—	MOTOR<exp.>
MPF	—	—	—	—	—	—
NCR DM-V (MS-BASIC)	—	—	—	—	—	—
NEW BRAIN	—	—	—	—	—	—
ORIC	—	—	—	—	—	—
SHARP MZ-700 (MZ-BASIC)	—	—	—	—	—	—
SINCLAIR QL	—	—	—	—	—	—
SPECTRAVIDEO	STICK(<n>)	STRIG(<n>)	ON STRIG GOSUB<nl.>[,...,<nl.>]	STRIG(<n>)ON/OFF/STOP	—	MOTOR ON/OFF
ZX-SPECTRUM	—	—	—	—	—	—

<n>: número de «port» o toma de entrada. <nl.>: número de línea inicial de una subrutina. <exp.>: expresión numérica.

ejecución de la zona en la que no se deseaban interrupciones. Este control se logra con la instrucción siguiente:

<Núm. de línea> STRIG (<exp.>) [ON/OFF/STOP]

Con la referida instrucción y mediante el número consignado en la zona <exp.>, se puede elegir el botón de disparo cuya detección con la instrucción ON STRIG GOSUB se desea activar, desactivar o memorizar. Para permitir la interrupción generada por un determinado botón de disparo se debe utilizar este comando en la forma: STRIG ( ) ON, con lo que al pulsar el botón afectado se pasará a ejecutar la subrutina cuyo número de línea se haya especificado con anterioridad en la instrucción ON STRIG GOSUB.

Si lo que se quiere es desactivar esa interrupción, será preciso hacer uso de STRIG ( ) OFF; con ello, el BASIC ignorará las pulsaciones del botón de disparo destinadas a generar una interrupción.

Al ejecutar STRIG ( ) STOP, la interrupción no se activará al pulsar el botón de disparo, si bien este hecho será anotado internamente, y en el mismo momento en el que se vuelva a activar el control mediante una instrucción STRIG ( ) ON, la interrupción tendrá lugar, aunque no se esté pulsando el botón de disparo en ese preciso momento.



```
 10 REM MANEJO DE UN JOYSTICK
 20 CLS:LET X=20:LET Y=10:LET X1=20:LET
    Y1=10
 25 REM BORRAR POSICION ANTERIOR
 30 PRINT AT (X1,Y1);" "
 35 REM IMPRIMIR NUEVA POSICION
 40 PRINT AT (X,Y);"*"
 50 LET X1=X:LET Y1=Y
 55 REM OBTENER CODIGO DE PALANCA
 60 LET A=STICK(1)
 65 REM JOYSTICK EN REPOSO
 70 IF A=0 THEN GOTO 60
 80 REM JOYSTICK EN ACCION
100 IF A=1 THEN LET Y=Y-1
110 IF A=2 THEN LET Y=Y-1;LET X=X+1
120 IF A=3 THEN LET X=X+1
130 IF A=4 THEN LET Y=Y+1:LET X=X+1
140 IF A=5 THEN LET Y=Y+1
150 IF A=6 THEN LET Y=Y+1:LET X=X-1
160 IF A=7 THEN LET X=X-1
170 IF A=8 THEN LET Y=Y-1:LET X=X-1
180 REM LIMITAR POSICION EN PANTALLA
190 IF X<2 OR X>37 THEN X=X1
200 IF Y<2 OR Y>21 THEN Y=Y1
210 GOTO 30
```

PROGRAMA 1: Programa ejemplo ilustrativo del comportamiento de la función STICK ( ). El objetivo es desplazar un asterisco a través de la pantalla bajo el control de un joystick.

El juego del «ping-pong»

En la ciencia informática el empleo de anglicismos está a la orden del día. Esto viene a cuento del nombre que recibe otro dispositivo periférico, bastante popular, conocido por "PADDLE" («raqueta de ping-pong»). Despista bastante este nombre al ver el aspecto externo de tal dispositivo, el cual consta de una caja con un mando giratorio que mueve una resistencia variable o «potenciómetro». El origen del nombre se debe a que era empleado por el primer videojuego que se comercializó (el PING-PONG de ATARI), para gobernar las raquetas que impedían que la pelota se colase por el fondo de la pantalla... De ahí que se quedara con el nombre de «raqueta».

El *paddle* es un periférico de entrada de datos, al igual que el *joystick*; pero a diferencia de éste, que sólo puede entregar ocho códigos, el *paddle* entrega más de doscientos códigos diferentes. Ello no significa que se utilice para elegir entre todas estas opciones, sino que su principal uso es para controlar fenómenos que varíen de forma continua entre dos valores extremos, como por ejemplo los comandos de sonido o la posición horizontal o vertical de una determinada figura representada en la pantalla. Su funcionamiento se basa en hacer variar una tensión de forma continua entre dos valores mediante el potenciómetro. Esta señal eléctrica es codificada en forma digital mediante un convertidor analógico-digital, a cuya salida se accede mediante la función "PADDLE" cuyo formato es el siguiente:

<Núm. de línea> X=PADDLE(<exp.>)

De nuevo, se observa que este formato es igual al de las funciones STICK y STRIG, por lo que no es preciso volver a comentar aquí lo dicho anteriormente para esas funciones. La única diferencia es que la función "PADDLE" proporciona como resultado un valor comprendido, normalmente, entre 0 y 255, aunque este rango depende como siempre de cada dialecto BASIC. Este valor se manipulará a través de alguna variable en la que será almacenado, o bien directamente utilizando la función "PADDLE" como argumento de otra función, como puede observarse en el ejemplo que sigue a estas líneas, el cual simula una pantalla de «TELE-SKETCH»:

```
10 SCREEN 3:REM MODO GRAFICO
20 PSET(PADDLE(1),PADDLE(2))
30 GOTO 10
```

Control del motor

El BASIC no sólo permite controlar periféricos de entrada como los vistos hasta ahora, también hay periféricos de entrada y salida o sólo salida que pueden ser gobernados a través de órdenes en BASIC. Así, por ejemplo, puede ser útil en ciertos casos la posibilidad de conectar y desconectar el motor de un casete de audio, para hacer avanzar automáticamente a la cinta hasta un punto determinado en el que se encuentre situada cierta información. Para esta misión el BASIC dispone de un comando de formato muy simple que permite conectar y desconectar a voluntad el motor del casete:

<Núm. de línea> MOTOR [ON]/[OFF]

Si se elige la opción MOTOR ON, se activa un relé cuyos contactos pueden conmutar una pequeña corriente (de alrededor de 150 mA), capaz de proporcionar energía para el movimiento de un motor de casete, con lo que éste será operativo. Para desconectar el motor habrá que ejecutar la orden opuesta: MOTOR OFF. La utilización de este comando no sólo se limita al gobierno del casete. Puesto que los contactos destinados al control remoto del casete están disponibles normalmente en algún conector externo, se pueden acoplar a él otros dispositivos; pero esto ya entra dentro del campo del bricolage electrónico ajeno al fin de esta obra.

Impresoras

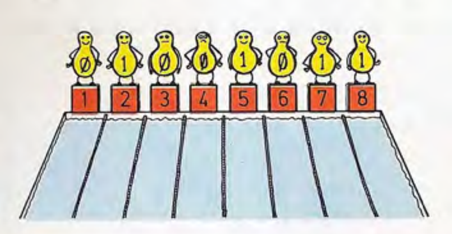
El periférico de escritura

Es posible que muchos usuarios de ordenadores personales no se hayan ni tan siquiera planteado la posibilidad de utilizar una impresora. Realmente, si el ordenador sólo se utiliza con programas de juegos o aplicaciones domésticas sencillas, no necesitará la colaboración de una de estas «máquinas de escribir inteligentes». No obstante, a medida que se adquiere una mayor experiencia en el uso del ordenador, se deseará obtener un mayor partido de sus ingentes posibilidades; ello conducirá a tener en cuenta el empleo de este útil periférico de salida.

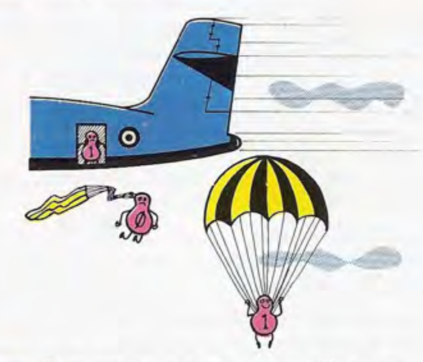
A la hora de confeccionar programas, la colaboración de una impresora resulta de indudable interés; con ella será posible guardar una copia escrita en papel de los listados. El hecho de disponer de copias parciales o totales del programa durante las fases de codificación y depuración resulta de gran ayuda. Su utilidad es extensiva a la obtención de copias impresas de los cálculos realizados por un determinado programa o, sencillamente, de los datos elaborados por un programa, por ejemplo, especializado en la gestión del gasto familiar.

Debido a la gran variedad de modelos que existen en el mercado, la elección por parte del usuario de la impresora adecuada a sus gustos y necesidades debe realizarse con sumo cuidado y conocimiento de causa. Para apoyar tal elección, hay que conocer las características más significativas que es preciso evaluar. Pero vayamos por partes, y empecemos por definir someramente qué es una impresora. Se trata, ni más ni menos, de un periférico de salida que transforma las señales eléctricas procedentes del ordenador en una forma legible para el ser humano; más concretamente, en un documento impreso en papel.

Como se desprende de esta definición, es el ordenador quien gobierna los movimientos de la impresora. La intervención del usuario tan sólo es necesaria para la correcta conexión de ambas máquinas, y para «dictar» al ordenador las instrucciones necesarias para que éste transfiera la información adecuada a la impresora.



En los sistemas de comunicación en formato paralelo, los ocho bits que conforman un byte parten y se transfieren simultáneamente desde el dispositivo emisor.

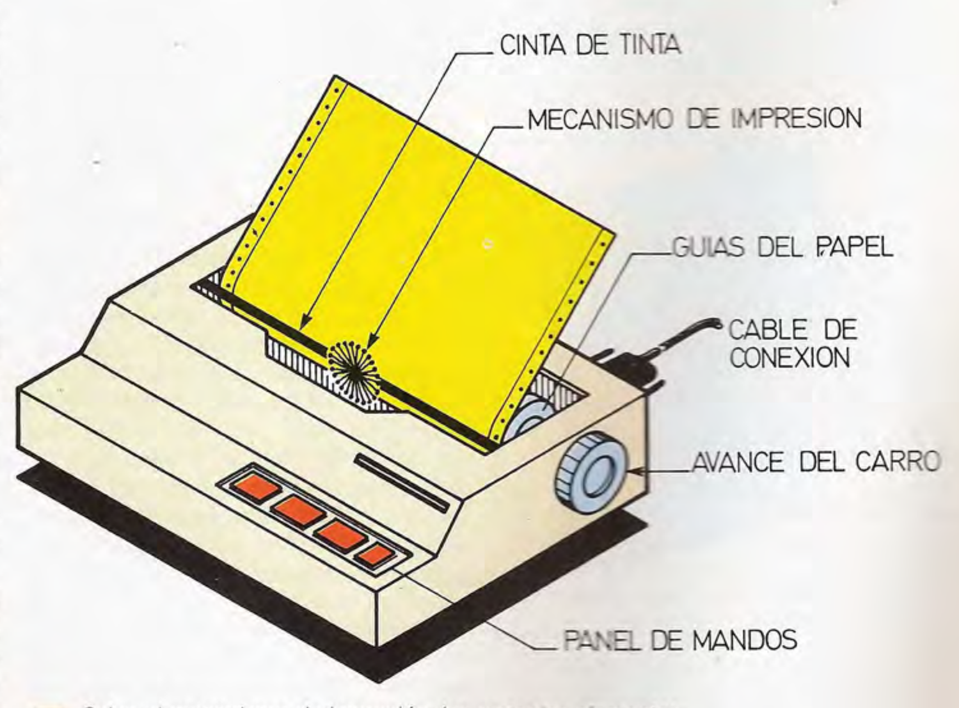


Si la comunicación se establece en formato serie, los bits son canalizados ordenadamente, uno tras otro, sobre una misma línea de transmisión.

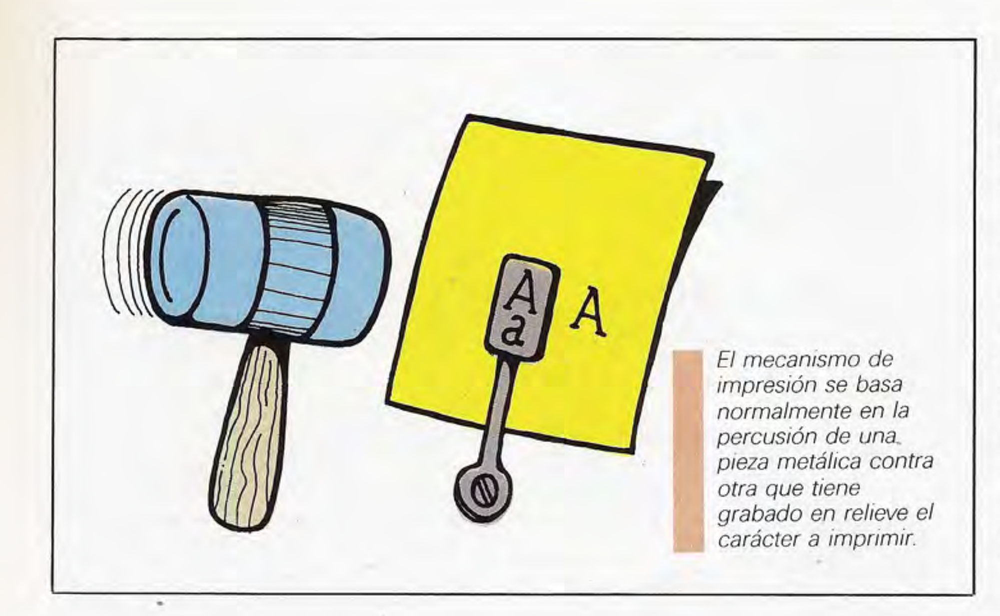
Tipos de impresora

La principal diferencia entre los distintos tipos de impresoras reside en la técnica empleada en sus cabezales de impresión; éste es, precisamente, el elemento que se encargará de plasmar los caracteres en el papel. Sus restantes elementos constitutivos son análogos; tan sólo muestran pequeñas diferencias en algunos aspectos tales como la forma de arrastre del papel, mandos de control, circuitos electrónicos de gobierno, conexiones con el exterior...

Son tres los tipos de impresoras más frecuentes en el ámbito de microordenador: las matriciales, las de margarita y las térmicas. La impresora matricial basa su funcionamiento en un juego de «agujas» metálicas, situadas en la cabeza de impresión, que forman una cuadrícula (matriz) con la cual se van componiendo los diferentes caracteres. Al mismo tiempo que la cabeza de impresión se va desplazando por el papel, las señales eléctricas que envía el ordenador se encargan de activar esas agujas según corresponda el carácter a imprimir. Las agujas golpean contra una cinta entintada, y ésta sobre el papel, di-



Salvo el mecanismo de impresión, los restantes elementos suelen ser comunes a la mayoría de las impresoras.



El mecanismo de impresión se basa normalmente en la percusión de una pieza metálica contra otra que tiene grabado en relieve el carácter a imprimir.

bujando cada aguja activada un punto de la superficie del mismo.

Mediante la adecuada combinación de estos puntos, se van creando los diversos caracteres que componen el texto. Con este mismo método, es posible imprimir también gráficos y dibujos; ello supone tan sólo codificar correctamente la información que debe activar a las distintas agujas de la matriz.

Las impresoras matriciales constituyen el tipo más corriente en el mundo de los ordenadores personales. Entre sus ventajas cabe citar una buena velocidad de impresión y una moderada economía; sin lugar a dudas, su sistema de impresión resulta sencillo en comparación con el de los restantes tipos de impresoras. Aunque también cabe hablar de inconvenientes; por ejemplo: son bastante ruidosas y su calidad de impresión suele ser reducida, debido a que las letras y números están formados por una conjunción de puntos estampados en el papel.

Las impresoras de chorro de tinta proyectan sobre el papel gotas de tinta en la proporción adecuada para «pintar» literalmente el carácter.

Cuando es necesario obtener copias de gran calidad y con una buena presentación, suele recurrirse a las impresoras de rueda de margarita; así llamadas por utilizar como mecanismo de impresión una rueda con pétalos, cuya forma recuerda a la mencionada flor, en cuyos extremos se encuentran situados los diferentes caracteres imprimibles.

El «modus operandi» de este tipo de impresoras consiste en el giro de la rueda de la margarita y la actuación sincronizada de un «martillo» metálico; éste golpea en cada caso al «pétalo» que transporta el carácter cuyo código corresponde con las señales eléctricas recibidas del ordenador. El martillo golpea entonces contra el pétalo, la cinta y el papel, dejando huella impresa del carácter en cuestión.

La calidad de impresión que se obtiene con este tipo de impresoras es comparable al de las mejores máquinas de escribir. Esta es su principal virtud, que la hace muy adecuada para aplicaciones tales como el tratamiento de textos, donde se requiere una gran calidad de letra. En muy pocos segundos es posible cambiar la margarita por otra con distinto alfabeto de caracteres, logrando así distintos tipos de impresión.

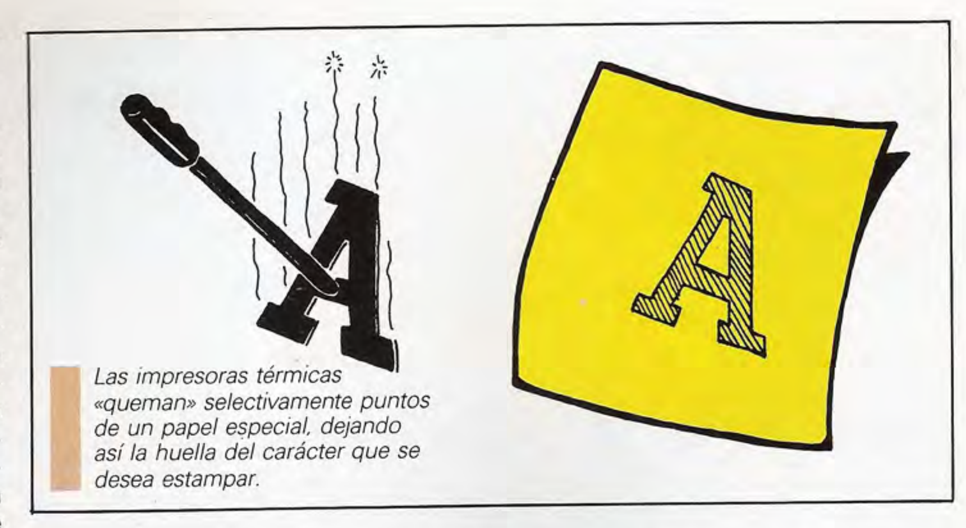
Como principales inconvenientes cabe señalar que las impresoras de margarita son bastante más lentas que las matriciales; por lo demás, con ellas no es posible crear gráficos o dibujos, pues haría falta una gran cantidad de ruedas diferentes para reproducir los innumerables matices de un dibujo. Otra desventaja es su precio, bastante más elevado que las matriciales.

El tercer tipo de impresoras, extendido en el campo de los microordenadores, lo constituyen las impresoras térmicas. En ellas, los caracteres se plasman por efecto del calor. Este es traspasado desde una cabeza de impresión especial, formada por resistencias eléctricas que se calientan al recibir una señal, a un papel sensible al calor, que se oscurece bajo su influencia. Este método de impresión cuenta con la ventaja de ser muy silencioso, además de bastante rápido y razonablemente económico. A estas ventajas se opone la nece-

Las impresoras de margarita deben su nombre a la similitud que presenta el elemento de impresión con dicha flor. El dispositivo consiste en una rueda en cuyos «pétalos» están situados los caracteres que es capaz de representar.

sidad de un papel especial, más caro que el normal, y la baja calidad de la impresión.

Existen otros muchos tipos de impresoras; si bien, los mencionados en las líneas anteriores son los más populares y accesibles para el usuario medio de un ordenador personal. Entre los restantes modelos, cabe citar aquí a las impresoras de chorro de tinta, que van «pintando» literalmente los caracteres por medio de gotas de tinta, lanzadas desde la cabeza de impresión y dirigidas mediante campos electromagnéticos. También hay que hacer mención a las impresoras de líneas, que escriben «línea a línea» en lugar de carácter a carácter, con lo que consiguen una mayor rapidez de impresión; y a las de láser, que ponen en práctica sofisticadas técnicas que conducen a un resultado muy veloz y de alta calidad.


Las impresoras térmicas «queman» selectivamente puntos de un papel especial, dejando así la huella del carácter que se desea estampar.

La impresora desde el BASIC

Las impresoras, al igual que los ordenadores, son máquinas incapaces por sí mismas de realizar alguna acción. De la misma manera que al ordenador hay que adiestrarle por medio de un programa, a la impresora también hay que indicarle las tareas que se desea que realice; pero en este caso es el mismo programa y, por lo tanto, el propio ordenador el que sirve de intermediario entre el usuario y la impresora. Las órdenes que determinarán el funcionamiento de la impresora adoptarán la forma de comandos BASIC.

¿Cómo se puede obtener una copia impresa del listado del programa? Como ya se sabe, el listado del programa se puede visualizar en pantalla o monitor mediante el comando LIST. En buena lógica, la orden adecuada para extraer el listado por impresora no debería diferir mucho de ésta; y así es en realidad. Para tal fin se emplea este mismo comando, aunque con ciertas variantes.

En efecto, es preciso indicar al ordenador, de una u otra forma, a través de qué dispositivo periférico de salida (pantalla, impresora...) se desea la emisión del listado del programa. Como quiera que, generalmente, esto se realiza por pantalla, si tan sólo se formula la orden LIST, el ordenador asumirá el que el listado debe visualizarse por pantalla. Para hacerlo a través de la impresora, será preciso indicar este hecho mediante el código correspondiente a este dispositivo que, generalmente, es la letra P (del inglés PRINTER: impresora) encerrada entre comillas. Así, el comando que en muchos dialectos BASIC permite obtener un listado por impresora presenta el siguiente formato:

<nl.> LIST "P",[<nl1.>—< nl2.>]

En el que la expresión <nl.> indica el número de la primera línea del programa a partir de la cual se desea obtener en listado, y <nl2.> precisa la última línea a listar.

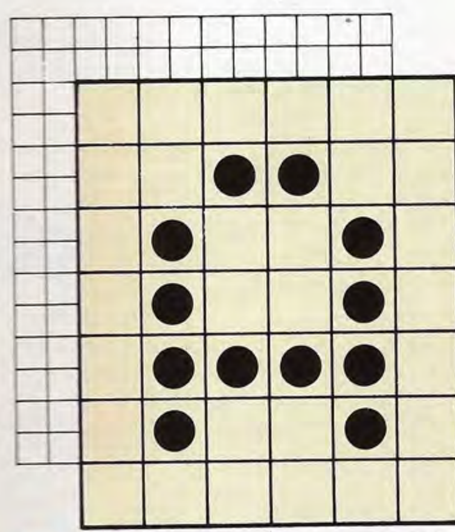
Por ejemplo, si en la memoria del ordenador se encuentra el siguiente programa:

```
10 REM EJEMPLO DE LISTADO
20 CLS
30 PRINT "PULSA UNA TECLA"
40 IF INKEY$="" THEN GOTO 40
50 PRINT "ESA TECLA NO VALE"
60 GOTO 30
```

Al ejecutar la orden LIST "P" se obtendrá un listado del programa completo por impresora; mientras que si la orden introducida es LIST "P", 20-50, el resultado en la impresora será el siguiente:

```
20 CLS
30 PRINT "PULSA UNA TECLA"
40 IF INKEY$="" THEN GOTO 40
50 PRINT "ESA TECLA NO VALE"
```

Así, pues, si el programa es extenso y se está alimentando a la impresora mediante hojas sueltas, se podrá hacer un cálculo del número de líneas de programa que caben en una hoja y, de esta forma, fraccionar el listado en varias hojas. De no hacerlo así, lo más normal es que al llegar al final del papel la impresora siga listando el programa con lo que, al poder avanzar el papel, el resultado sería un inmenso borrón indescifrable en la última línea impresa.

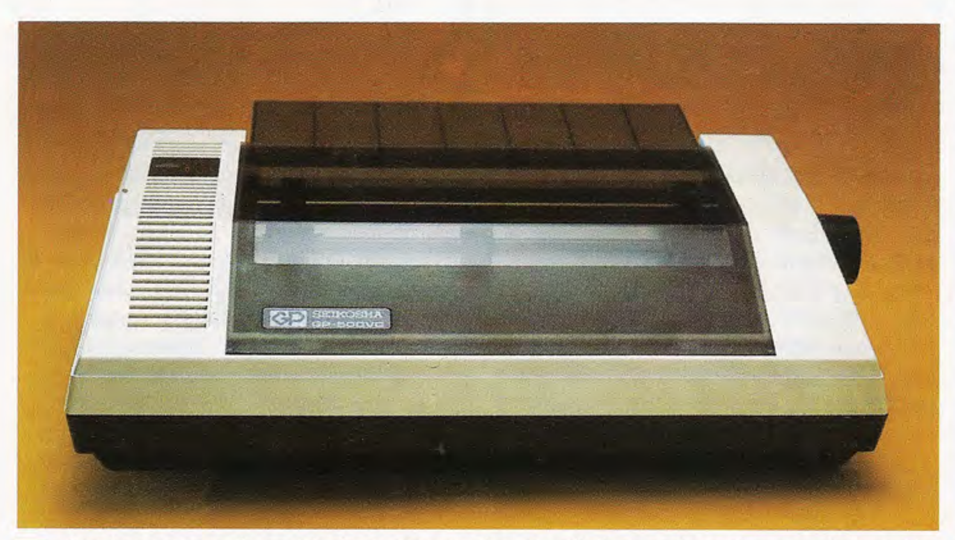

La formación de los caracteres en las impresoras matriciales tiene lugar por medio de una serie de puntos. Estos son estampados sobre el papel con unos minúsculos martillos (las agujas) que presionan a la cinta entintada sobre el papel al ser activados por señales procedentes del ordenador.

LIST "P"

Obtiene un listado del programa por impresora; permite especificar los números de líneas a imprimir.

Formato: LIST "P", <nl1>—<nl2>

Ejemplos:
```
10 LIST "P"
20 LIST "P", 10—1120
30 LIST "P", —100
```



Las impresoras de matriz de puntos suelen constituir una alternativa muy frecuente para el usuario de un ordenador de tipo doméstico o personal. Su precio moderado y su aceptable calidad abogan por tal elección.

Por ejemplo, otro de los comandos frecuentes para el mismo cometido es LLIST, cuyo formato y funcionamiento coinciden con lo indicado en el caso anterior. En otros traductores BASIC, es necesario «abrir» un canal de comunicación entre el ordenador y la impresora, y debiendo indicar este hecho a continuación del comando LIST, por medio de un número de identificación del canal previamente abierto. Este último caso se discribirá ampliamente en capítulos sucesivos.

Una vez que se establezca la correspondiente comunicación entre ordenador e impresora, puede ocurrir que el primer listado que se obtenga sea desastroso: con la presencia de líneas en blanco entre cada dos líneas del programa, o sin que se produzca ningún avance del papel, con lo que el resultado será una preciosa línea del color negro más oscuro que contiene, ella sola, todo el listado del programa. Esto puede ser debido a la diferencia existente entre los caracteres de «retorno de carro» y de «salto de línea» que le son enviados a la impresora por el ordenador. En efecto, el ordenador enviará un retorno de carro al final de cada línea de texto que hará retroceder la cabeza de impresión al comienzo de la línea y, tras ello, mandará un carácter de salto de línea que hará avanzar el papel.

El carácter de salto de línea es gene-

La indicación de los números de línea inicial y final también hace posible obtener listados de sólo algunos trozos interesantes del programa, como subrutinas de uso general o algún segmento especial del mismo.

El lenguaje BASIC no está demasiado estandarizado en este aspecto. Son muchos los intérpretes BASIC que no aceptan el referido comando, siendo necesario el uso de otros comandos alternativos.



Algunos fabricantes ofrecen para sus equipos una alternativa que combina la función de impresora con la de plotter o trazador gráfico. Este tipo de periféricos de escritura dibuja los caracteres y las líneas del trazado por medio de «plumas» o bolígrafos de diversos colores.

rado automáticamente por el ordenador, o por la misma impresora, con lo que en el caso de concurrir ambos o ninguno de los dos, pueden llegar a producirse los desastrosos resultados descritos. La solución estriba en una lectura cuidadosa de los manuales de usuario, tanto de la impresora como del ordenador en cada caso específico.

Una vez que ya sabe obtener listados por impresora, el usuario estará ansiosos de conocer la forma de imprimir los mensajes que desee o los resultados de sus propios programas. Pues bien, la forma de lograrlo es tan sencilla como la utilizada para los listados. Como ya se adivina, guarda una estrecha relación con los comandos adecuados para ordenar esa misma función sobre la pantalla del TV.

Mientras que para imprimir caracteres sobre la pantalla se utiliza el comando PRINT, para que la salida se produzca por la impresora el comando indicado suele ser LPRINT, cuyo formato más general es el siguiente:

<nl,> LPRINT <exp.>

La expresión <exp.> puede ser una cadena de caracteres, una variable, un número directamente o una expresión cuyo resultado se calcula e imprime en papel. Así, por ejemplo, las siguientes líneas:

```
10 LPRINT "HOLA"
20 LET N=1234
30 LPRINT "N=";N
40 LPRINT
50 LPRINT 5678
60 LPRINT 4*6
70 END
█
```

harán que la impresora escriba el texto que sigue:

```
HOLA
N=1234

5678
24
```

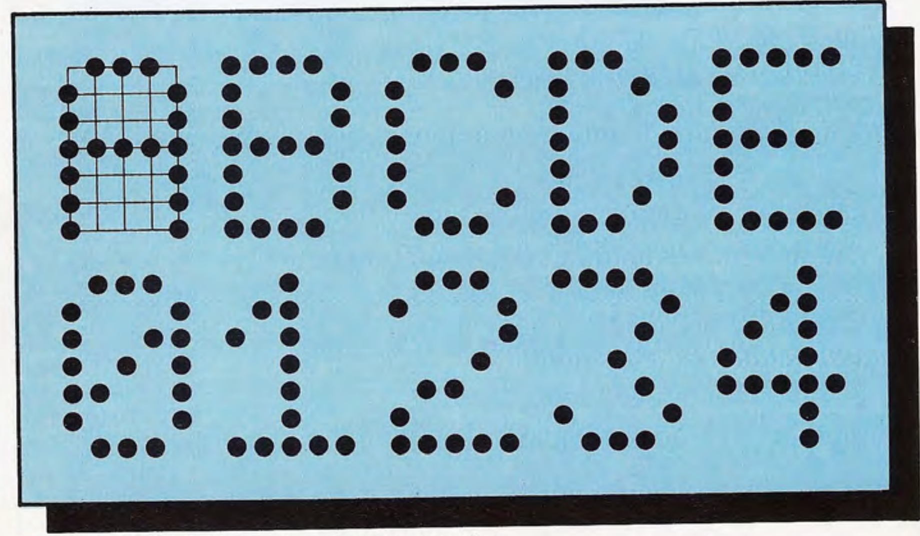
Como se puede observar en la línea 30 del ejemplo anterior, es posible utilizar los separadores típicos de la sentencia PRINT, como son la coma y el

LPRINT

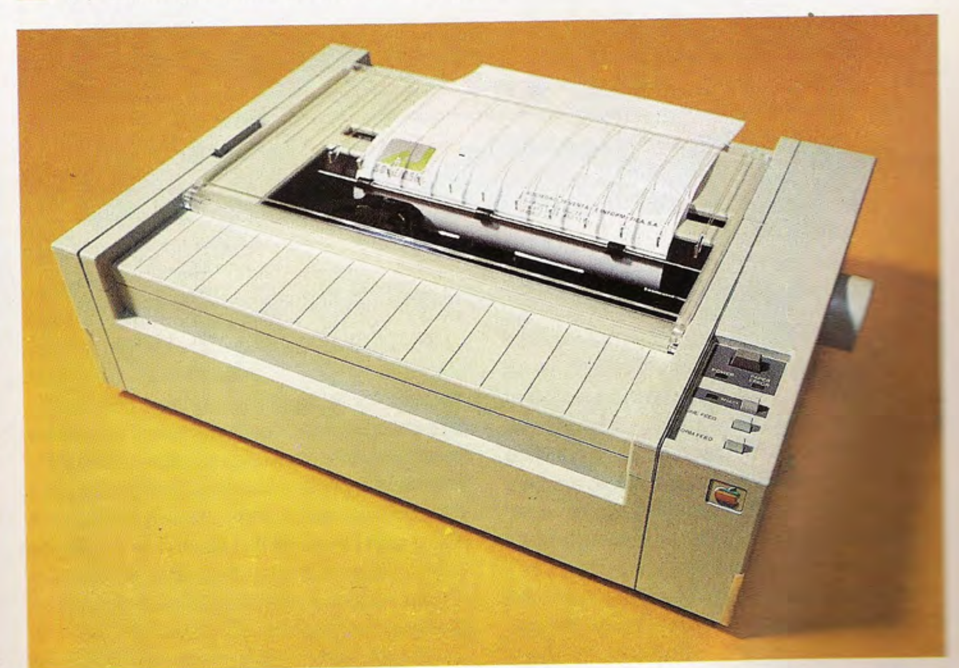
Imprime el contenido de variables, datos literales o expresiones en general.

Formato: LPRINT <exp.>{[;][,]}[<exp.>]

Ejemplos: 10 LPRINT NUM;
20 LPRINT "ADIOS"
30 LPRINT 5 * 3



Composición típica de caracteres creados por una impresora matricial de 7×5 puntos.



En la fotografía aparece la impresora ImageWriter de la firma Apple. Esta es una impresora de matriz de puntos facultada para la impresión de muy diversos alfabetos (fuentes) de caracteres.

LPRINT AT

Lista valores por impresora, permitiendo la tabulación en columnas.

Formato: LPRINT AT x,y; <exp.>
Ejemplos: 10 LPRINT AT 0,0; "HOLA"
20 LPRINT AT 10.0; "ADIOS"

```
 10 CLS
 20 PRINT "TALONARIO DE CHEQUES"
 30 PRINT
 40 DIM NOMBRE$(20),PESETAS$(50),FECHA$(20),MES$(15)
 50 INPUT "CANTIDAD EN NUMERO";C
 60 INPUT "DESTINATARIO";NOMBRE$
 70 INPUT "CANTIDAD EN LETRA";FECHA$
 80 INPUT "FECHA EN LETRA";FECHA$
 90 INPUT "MES";MES$
100 INPUT "  DE  198";YEAR
110 LET YEAR=YEAR+1980
120 LPRINT "
";CHR$(15);"PTA  *";C;"  pts.*";CHR$(14)
130 LPRINT
140 LPRINT CHR$(15);"Paguese a  ";NOMBRE$
150 LPRINT "Pesetas  ";PESETAS$
160 LPRINT CHR$(14);"                              ";CHR$(15);FECHA$;"
de  ";MES$;"            de ";YEAR;CHR$(14)
180 FOR I=1 TO 5:LPRINT  :NEXT I
```

Listado del programa "Talonarios de cheques".

punto y coma. Ambos producirán los resultados ya conocidos, aunque, en este caso, sobre la hoja de papel.

La orden LPRINT, sin más, produce en la impresora una línea completamente en blanco, ya que en este caso el ordenador únicamente envía al referido periférico los caracteres de retorno de carro y de salto de línea. Por otro lado, es posible utilizar en la mayoría de los casos las características de tabulación propias del comando PRINT AT x,y. Aunque con la salvedad que sólo funcionará el posicionamiento de la cabeza de impresión en la columna indicada, y no será posible su colocación en otra línea del papel distinta a la actual; ello se debe a que, en general, las impresoras no cuentan con mecanismos para el retroceso del papel.

Veamos un nuevo programa a título de ejemplo. Su cometido es escribir aleatoriamente los signos de una imaginaria quiniela futbolística.

```
10 LPRINT "QUINIELA":LPRINT
20 FOR I=1 TO 14
30 LET A=INT(RND(0)*100)
40 IF A<50 THEN LPRINT "1"
50 IF A>15 AND A<=45 THEN LPRINT "X"
60 IF A<=15 THEN LPRINT " 2"
70 NEXT I
```

Partiendo de unas determinadas probabilidades (en porcentaje) y de un número aleatorio, entre 0 y 100, generado por el ordenador, se van obteniendo los diferentes signos 1, X ó 2. Signos que aparecerán impresos en el papel en su correspondiente columna; un posible resultado es el que aparece a continuación:

```
QUINIELA

1
1
1
 X
1
  2
 X
1
1
 X
 X
1
  2
  2
```

Queda a la atención del lector la posible —y muy sencilla— modificación de este programa para que sea capaz de escribir sobre un boleto original de apuestas en lugar de sobre una hoja de papel convencional.

Caracteres de control

Dentro del repertorio de caracteres del código ASCII, empleado normalmente por la gran mayoría de ordenadores para la representación interna de los caracteres, existen una serie de códigos dedicados exclusivamente al control de diversos periféricos y otros menesteres; códigos que no plasman un resultado escrito ni sobre la pantalla ni sobre el papel. Los referidos códigos se utilizan con distinto objetivo en los diversos modelos de impresoras, aunque siempre para activar sus posibles «efectos especiales» de impresión.



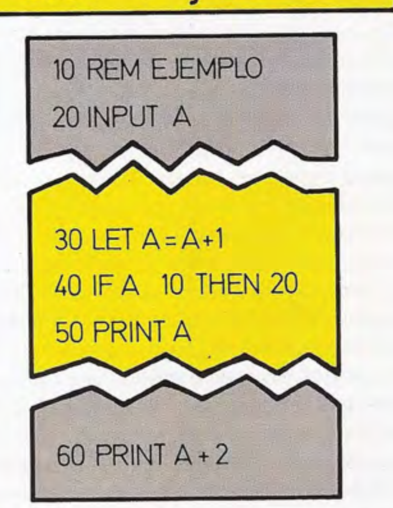
El comando LIST «P» permite obtener listados de programas a través de la impresora en un soporte permanente de papel.

TABLA DE CONVERSION		
Ordenador	LLIST	LPRINT
	LLIST [<n1>-<n2>]	LPRINT <exp.>
AMSTRAD	LIST [<n1>-<n2>],#8	PRINT #8,<exp.>
APPLE II (APPLESOFT)	—	—
APRICOT (M-BASIC)	LLIST [<n1>-< n2>]	LPRINT <exp.>
ATARI	LIST "P",[<n1>-<n2>]	LPRINT <exp.>
CBM64	—	—
DRAGON	LLIST [<n1>-<n2>]	PRINT#-2<exp. >
EQUIPOS MSX	LLIST [<n1>-<n2>]	LPRINT <exp.>
HP-150	LLIST [<n1>-<n2>]	LPRINT <exp.>
IBM PC	LLIST [<n1>-<n2>]	LPRINT <exp.>
MPF	—	—
NCR DM-V (MS-BASIC)	LLIST [<n1>-< n2>]	LPRINT <exp.>
NEW BRAIN	—	—
ORIC	LLIST [<n1>-<n2>]	LPRINT <exp.>
SHARP MZ-700 (MZ-BASIC)	LIST/P [<n1>-< n2>]	PRINT/P <exp.>
SINCLAIR QL	—	—
SPECTRAVIDEO	LLIST [<n1>-<n2>]	LPRINT <exp.>
ZX-SPECTRUM	LLIST [<n1>-<n2>]	LPRINT <exp.>

Los caracteres de control serán enviados a la impresora, desde el ordenador, de diferentes formas. Los métodos más comunes consisten en utilizar la función CHR$() como argumento de un comando LPRINT, o bien mediante la generación de algún carácter alternativo pulsando la tecla CONTROL o ESCAPE y otra simultáneamente.

Entre los «efectos especiales» que se pueden conseguir en la mayoría de las impresoras comerciales, cabe mencionar los siguientes: espaciado proporcional, que consiste en asignar menor espacio en el papel a los caracteres más estrechos como la «i» y más espacio a los más anchos como la «w»; selección del tipo de letra, en las impresoras matriciales entre tipos tales como elite, pica, elite alargada, pica condensada, cursiva, cursiva enfatizada, etc.; doble percusión del martillo, que imprime dos veces susperpuestas un mismo carácter, logrando una mejor calidad de letra; activación y desactivación del subrayado del texto a escribir; selección de ciertos caracteres especiales, como la «eñe» española o los signos de diéresis o acentos; modificación de los patrones de caracteres de las impresoras matriciales para así poder crear gráficos y diagramas...

Debido a la casi nula estandarización de los caracteres de control empleados por las impresoras, es imposible relatar aquí la función de los mismos. Mientras en un modelo, el código 24 se emplea, por ejemplo, para activar el subrayado, en otro modelo puede no producir efec-



Recurriendo a la opción de indicar los números de línea detrás del comando LIST "P", es posible obtener listados parciales de programa.

PTS #50.000 pts.#

Paguese a JOSE PEREZ PEREZ

Pesetas CINCUENTA MIL

TREINTA de MAYO de 1985

Un posible resultado impreso derivado de la ejecución del programa «talonario de cheques».

to alguno, o bien desencadenar una acción completamente distinta.

Para generalizar la cuestión, en el siguiente ejemplo (ver cuadro adjunto), cuya misión es editar cheques impresos, se presupone que el carácter CHR$(15) activa el efecto de subrayado mientras que CHR$(14) lo desactiva. Ambos códigos deben ser modificados en función del modelo de impresora que se utilice.

Un posible resultado de su ejecución es el que aparece en el cuadro adjunto.

Tomando como punto de partida el referido programa, el lector puede practicar el manejo de su impresora y acondicionar el programa para que sea capaz de rellenar cheques auténticos. Aunque, desde luego, la firma siempre debe partir del propio puño y letra del titular de la cuenta.

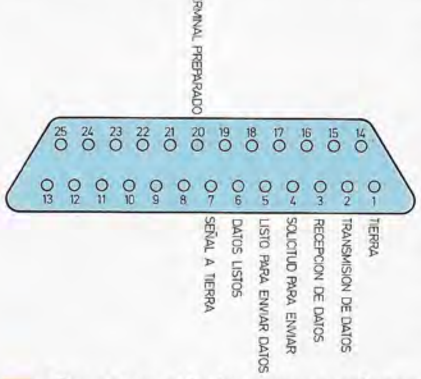
Medios de comunicación

Los dispositivos periféricos que rodean al ordenador y permiten su comunicación con el mundo exterior, exigen la presencia de un medio especializado en el diálogo de entrada o salida, medio que debe facilitar el trasvase de las órdenes de trabajo y de la propia información.
Los dispositivos que permiten establecer tal comunicación obedecen al apelativo genérico de «interfaces». Esta palabra engloba tanto a los circuitos electrónicos como al software que lo gobierna.
Internamente, los ordenadores poseen los denominados «buses», formados por varios conductores en paralelo que transportan información binaria. Dicha información corresponde a los datos en proceso, a órdenes de control y a direcciones de memoria. Mediante estas señales, la CPU «sabe» cuándo debe enviar cierta información a un periférico o recibirla de éste, y también conoce si un periférico determinado se encuentra disponible o no. El conocimiento de estas situaciones se obtiene en base al adecuado intercambio de señales de control; tal intercambio se canaliza a través de los interfaces que unen al ordenador con los periféricos, y respetando un determinado protocolo de comunicación previamente establecido. Normalmente, la comunicación entre el ordenador y la mayoría de los periféricos se establece en el llamado «formato paralelo»: a través de un conjunto de cables cada uno de los cuales transporta una señal binaria. Por extensión, los interfaces en los que se apoya este tipo de comunicación se denominan «interfaces en paralelo», y realizan el trasvase de información byte a byte. El tipo de interface más conocido y utilizado en el terreno de la comunicación con impresoras en formato paralelo es el llamado «CENTRONICS», en razón a la firma que la creó.

Es indudable que además de las líneas de datos son necesarias toda una serie de señales de control. En el caso del interface CENTRONICS, las señales presentes en el conector de comunicación son las siguientes:

ADK: disposición para aceptar datos.
GND: masa de referencia o «tierra» para el ordenador y periférico.
STROBE: señal del ordenador al periférico que revela que la información está ya preparada para ser leída.
BUSY: señal de periférico ocupado.
D0 a D7: ocho líneas de bit para la transferencia de los datos.

Aunque este tipo de interface es el más utilizado para la comunicación con impresoras, la estandarización del mismo no es perfecta.
Es fácil tropezarse con multitud de conectores diferentes



Conector estándar correspondiente a un sistema de comunicación en formato serie según la norma RS/232C. En el gráfico se indica el cometido de los terminales más relevantes.

que pretenden realizar el mismo tipo de unión, si bien, el problema no pasa generalmente de encontrar el cable adecuado a cada caso. Cuando los dispositivos a conectar se encuentran a considerable distancia física y su bus de datos incluye un gran número de señales, la conexión en paralelo no es recomendable. En tal caso se opta por la comunicación en «formato serie». En la comunicación en «serie», se van enviando por un único cable, y uno detrás de otro, los bits que constituyen la información a tansmitir. Aunque no se den las condiciones de lejanía antes indicadas, es muy frecuente encontrar ordenadores que disponen de «interfaces de tipo serie» para la comunicación con los dispositivos periféricos.
El tipo de interface serie más frecuente es el que responde a las siglas RS/232. Su definición a nivel teórico está bastante estandarizada por lo que respecta a los niveles de las señales eléctricas y a la asignación de los terminales de conexión. Se suele utilizar un conector «miniatura D» de 25 contactos, de los cuales el número 2 se emplea para transmitir los datos del ordenador al periférico, el número 3 para recibir los datos procedentes del periférico y el número 7 como masa de referencia o común de ambas señales.
Los dispositivos de transmisión y de recepción deben ser escrupulosamente ajustados en cuanto a velocidad de transmisión y al formato de la información enviada para que la transferencia se produzca correctamente.
Además de los dos citados, existen otros muchos tipos de interfaces; si bien, suelen ser específicos de un determinado modelo o gama de ordenadores.
Generalmente, éstos sólo permiten la transmisión de datos en un sentido, tal es el caso de los interfaces para joystick, para monitores de televisión, para entradas analógicas con fines de medición, para el control del motor de un magnetófono a casetes...

Indice temático

Macrolenguajes gráficos

Otras herramientas gráficas

Introducción al sonido

Los sonidos del BASIC

Del BASIC al código máquina